UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Registo de Dados Indelével

Cristiano Ramos Iria

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA

Especialização em Arquitetura, Sistemas e Redes de Computadores

2013

UNIVERSIDADE DE LISBOA

Faculdade de Ciências

Departamento de Informática

Registo de Dados Indelével

Cristiano Ramos Iria

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA

Especialização em Arquitetura, Sistemas e Redes de Computadores

Dissertação orientada pelo Prof. Doutor Paulo Jorge Esteves Veríssimo

2013

Agradecimentos

Ao longo dos anos como estudante da Faculdade de Ciências da Universidade de Lisboa, muitas pessoas fizeram parte deste percurso académico, contudo algumas assumiram um papel preponderante na minha vida, tanto profissional como pessoal.

Começo por agradecer ao Professor Doutor Paulo Jorge Esteves Veríssimo, Professor e orientador, o acompanhamento, clareza e suporte no presente trabalho e ainda pela possibilidade de integrar o grupo de investigação *Navigators*[1].

De seguida, menciono o prazer em ter conhecido alguns investigadores e professores da Faculdade de Ciências. O seu trabalho é notável e devia motivar todos os alunos desta instituição.

Às pessoas que me ajudaram durante todo o ano, ou na fase final desta dissertação, quero deixar o meu profundo agradecimento. Entre essas pessoas estão a Sara Canhoto, a Alexandra Rodrigues, o Hugo Sousa, o Diego Kreutz e a Madalena Vaz da Silva.

Agradeço a todos os meus colegas e amigos de Licenciatura e Mestrado, pelo conhecimento partilhado e companheirismo. Ao Lab 25, local de trabalho dos investigadores juniores do grupo *Navigators*, onde estive inserido, um especial obrigado e, partilhando a opinião do Hugo Sousa, continuo a achar que todos juntos nenhuma outra empresa teria hipótese.

Não menos importante, queria agradecer aos projetos que financiaram esta dissertação: o FDISAR - Fundo de Desenvolvimento para a Investigação em Sistemas Adaptativos e Resilientes, e o TRONE - *Trustworthy and Resilient Operations in a Network Environment*.

Longe de serem os últimos, o maior obrigado à minha família.

[1] http://www.navigators.di.fc.ul.pt/

Aos meus pais, irmã e Sara.

Resumo

A análise forense de computadores lida com a recolha de provas digitais de vários tipos de dispositivos eletrónicos. Esta ciência está diretamente ligada à captura, análise e reconstrução de atividades do sistema a fim de averiguar, *a posteriori*, o método do atacante ou uma possível avaria.

Os *logs* são uma parte importante de qualquer sistema computacional seguro, uma vez que registam a sua atividade. Com esse registo temos uma boa visão de todo sistema. Quando a última barreira de segurança é ultrapassada e ocorre um ataque ou há uma falha de um componente, os registos são um dos poucos recursos para entender o que se sucedeu.

Por outro lado, os *logs* encontram-se entre os alvos preferidos de alguém que quer penetrar num sistema, por dois motivos: em primeiro lugar, os *logs* têm pistas importantes sobre o sistema em causa e a sua segurança. Por outro lado, ao adulterar-los, é possível impedir que um administrador veja a sua atividade no sistema [33].

O objetivo deste trabalho é criar um sistema de *logs* centralizado para sistemas distribuídos, utilizando um modelo de faltas híbrido. Visto do exterior, o sistema é composto apenas por uma máquina, contudo no seu interior está uma estrutura complexa, replicada e tolerante a faltas. Este sistema garante segurança para os *logs* em trânsito e em repouso. Por forma a que o sistema permaneça correto a longo termo, são utilizadas técnicas como a recuperação pró-ativa e reativa e para não se perderem os *logs* já armazenados, utiliza-se um disco *Write Once Read Many* (WORM).

Para que a informação dos *logs* permaneça secreta, o sistema perde o controlo sobre o conteúdo dos *logs*, isto é, quando um *log* dá entrada no sistema é cifrado com recurso a um esquema de chave pública e só o detentor da chave privada poderá aceder ao seu conteúdo. Normalmente, o detentor dessa chave privada é o administrador ou uma entidade de auditoria.

Palavras-chave: *logs* seguros, *logging*, arquitetura distribuída de *log*, integridade de *logs*, deteção e tolerância de intrusões

Abstract

Forensic analysis of computers deals with the collection of digital evidence of several types of electronic devices. This science is directly linked to the capture, analysis, and reconstruction of system activities which investigate, *a posteriori*, the hacker's method or a possible system failure.

Logs are an important part of any secure computer system because they record the activity performed by and in the system, allowing one to obtain a good view of the whole system and its individual parts. For instance, when the last barrier of the system has been broken and an attack has happened or if there has been a failure in one component of a large-scale system, logs are one of the few resources through which one can know what occurred.

Logs are also one of the most wanted targets for someone who wants to penetrate in a system for two reasons: first, logs have important clues about the system and their safety; second, to prevent an administrator to see their activity on the system [33].

The objective of this work is to create a centralized logging system for distributed systems using a hybrid fault model. From an outsider's point of view, the system is just a machine, however in its inside there is a complex, replicated and fault tolerant structure. This structure provides a robust log system which secures the logs in transit and at rest. To ensure that the system keeps its long-term correctness it may be used proactive and reactive recovery. To avoid losing logs it will be used a Write Once Read Many (WORM) disk.

In order to guarantee that the logs' information remains secret, the system loses control over the content of the logs, i.e., when a log enters the system it is encrypted using a public key and only the private key owner can access its content. Usually, the owner of this private key is the administrator or an audit entity.

Keywords: secure logging, logging, distributed log arquitecture, log integrity, intrusion detection and tolerance

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Com a Internet, pudemos assistir aos ataques maliciosos, que deram origem a uma nova ciência na área da segurança informática, a chamada análise forense. Esta lida com a recolha de provas digitais de vários tipos de dispositivos eletrónicos, como computadores pessoais, encaminhadores, servidores, entre outros. Mais concretamente, esta ciência está ligada à captura, análise e reconstrução de atividades do sistema a fim de averiguar, *a posteriori*, como e quando é que um dispositivo foi comprometido; ou a identificar uma possível avaria do sistema [19]. A análise forense utiliza rastos de atividade, normalmente registados ou encontrados na forma de *logs*.

Um *log* é um registo de um evento que ocorreu num sistema ou rede e é parte importante de qualquer sistema computacional que necessita de elevados níveis de segurança, uma vez que, estes registam a atividade desempenhada pelo e no sistema, tal como, atividade de utilizadores, execuções de programas, utilização dos recursos do sistema, entre outras. Quando surgiram, eram utilizados para resolução de problemas mas, atualmente, servem muitos propósitos que vão desde a otimização de sistemas, registo de ações de utilizadores até à investigação de atividade maliciosa [15]. Os *logs* oferecem uma boa visão do estado passado/presente de qualquer sistema [16]. Quando a última barreira de segurança do sistema é ultrapassada e ocorre um ataque ou há uma falha de um componente de um sistema, os *logs* são recursos importantes para que seja possível entender o que se sucedeu.

Nas atuais infraestruturas informáticas, é de extrema importância ter um serviço de *logs* seguro. Seria interessante ter um serviço de *logs* seguro que pudesse ser instalado em qualquer sistema informático sem grandes dificuldades e que oferecesse segurança e resiliência. Assim, surgiu a ideia de desenvolver uma "caixa preta" que disponibiliza um

serviço de *logs* seguro.

No sistema existem duas grandes entidades: produtores e coletores de registos (servidor de *logs*). Os produtores de *logs* são dispositivos de rede, máquinas pessoais, servidores Web, entre outros, que produzem eventos *log*; os coletores de *logs* recebem esses eventos e registam-nos em ficheiros *log*.

A integridade dos *logs* é imperativa e significa que nenhuma entrada destes é alterada ou apagada. Os registos podem ser utilizados como prova digital e/ou material de apoio aos administradores de um sistema para identificar um intruso ou uma falha. Se os *logs* forem alterados de forma não detetável, o problema pode permanecer incógnito para o administrador durante um longo período de tempo, ou poderão ser utilizadas provas digitais falsas.

Com baixa disponibilidade, o sistema de *logs* não tem utilidade prática. Coloque-se a hipótese de um sistema se encontrar sob ataque e de que, durante o mesmo, não tem capacidade para processar todos os eventos, perdendo inclusive alguns. Na posterior análise forense do ataque, a totalidade da atividade do sistema, quer durante o ataque, quer os momentos que o antecederam, poderá não ter sido registada. Na ausência desta informação, será praticamente impossível compreender como foi levado a cabo o ataque. Desta forma, o desempenho e disponibilidade do sistema são de extrema importância para os fins a que este se destinam.

A confidencialidade é um ponto importante. Não é desejável que esta informação passe pelos servidores ou seja guardada sem algum tipo de proteção. Assim, é aconselhável que apenas uma entidade autorizada possa aceder ao conteúdo dos *logs*, nomeadamente, uma equipa de análise forense. Portanto, os *logs* devem ser cifrados com uma chave pública dessa entidade e só ela os poderá decifrar. Pode ainda considerar-se um cenário onde o sistema em causa é de alta segurança. Neste caso, para aceder ao conteúdo dos *logs* seria necessário combinar chaves de várias entidades (criptografia de limiar[24]).

Deste modo, ficam estabelecidas as propriedades que o sistema procura garantir:

- Integridade - uma vez a entrada escrita no ficheiro *log*, não poderá ser apagada ou modificada;
- Disponibilidade - o sistema deve ter alta disponibilidade para que não se percam eventos dos produtores e, consequentemente, a informação que continham;
- Confidencialidade - o sistema não tem controlo sobre o conteúdo de um evento que

entra no sistema. Só uma entidade autorizada, cuja chave foi utilizada para cifrar o *log*, poderá aceder ao seu conteúdo.

1.1 Motivação

Embora muitas organizações e utilizadores domésticos recorram a produtos de segurança como *firewalls*, sistemas de deteção e proteção contra intrusões, ou mesmo sistemas operativos seguros, os seus sistemas continuam a ser suscetíveis a ataques maliciosos, pois não existem soluções de segurança perfeitas.

Um dos pontos mais importantes na manutenção de um ambiente seguro é saber o que se passa realmente nesse ambiente [33]. Existem várias formas de o saber, uma delas é através do uso engenhoso e cuidadoso dos *logs*. Assim, os *logs* têm um papel importante na recolha de indícios sobre as atividades que o sistema realizou ou realizadas no sistema. Para encontrar as causas e os efeitos de uma intrusão na rede ou de um ataque, pode ser necessário instalar um sistema de *logs* que recolha os dados da rede ou dos dispositivos ligados à mesma. Esses dados recolhidos serão úteis para o administrador, que tenta repor a normalidade; ou para um auditor que examina o sistema com o objetivo de encontrar a causa do problema.

Os *logs* são também os alvos preferenciais durante uma tentativa de penetrar num sistema, o que ocorre essencialmente por duas razões. Em primeiro lugar, pelo facto dos *logs* de um sistema conterem, muitas vezes, pistas importantes sobre o sistema em causa e a sua segurança. Sendo assim, o acesso aos registos tem como primeiro objetivo a recolha do máximo de informação útil possível, relativa ao sistema em questão. A segunda razão consiste no facto de, se houver efetivamente uma penetração no sistema, os *logs* permitirem denunciar a atividade do atacante. Deste modo, a forma mais simples de impedir a deteção e consequente expulsão do atacante pelo administrador, é a adulteração de *logs*, por forma a que o administrador apenas observe o que seria de esperar durante a atividade normal do sistema.

Estes registos são a fonte primária para a investigação após um incidente ou para recuperação de um sistema. Podem ser utilizados como prova em tribunal[6] e como material de apoio aos administradores de um sistema para identificar um intruso (i.e., para resolver este problema, o administrador pode utilizar ferramentas de deteção de intrusões com recurso a monitorização e análise de *logs*). Em ambos os casos a integridade dos *logs* é fundamental.

O sistema vem colmatar as dificuldades dos sistemas de *logs*, i.e., a proteção dos *logs* em repouso. Os trabalhos até então desenvolvidos que, garantem proteção aos registos em repouso, apenas detetam adulterações nos *logs* [22, 16, 2]. Contudo, isso não é suficiente e, como tal, o sistema proposto garante proteção contra eliminação dos *logs*. Apenas uma entidade autorizada, por exemplo, o administrador, terá acesso aos registos.

1.2 Objetivos

Para facilitar o processo de tornar o sistema resiliente, o objetivo é criar uma solução centralizada de *logs* para ser integrada em sistemas distribuídos.

O sistema deve ter visto como uma "caixa preta", i.e., sem estrutura visível e com uma única e simples interface disponível para o ambiente exterior. Essa caixa tem como objetivo garantir a segurança dos *logs* em trânsito e em repouso.

Outro objetivo do sistema é eliminar a sobrecarga de esquemas criptográficos no armazenamento, uma vez que nenhum esquema criptográfico pode ser utilizado para prevenir a eliminação de entradas nos *logs*. Os esquemas criptográficos utilizados em outros trabalhos apenas permitem detetar adulterações dos registos, portanto não protegem contra eliminação do disco. Para resolver esse problema pode ser utilizado um disco *Write Once Read Many* (WORM) na zona mais segura do sistema (armazenamento) [22].

O servidor de *logs* deve ser robusto e resiliente, por outras palavras, resistir a ataques e intrusões. Para tal, o servidor remoto será replicado, terá uma arquitetura robusta e recuperação pró-ativa e reativa. Mesmo assim, uma vez que o servidor esteja comprometido, os atacantes não conseguirão apagar os *logs* anteriores, devido à área segura de armazenamento com disco WORM, mas conseguirão evitar que mais *logs* sejam armazenados. Embora não se consiga evitar ataques de negação de serviço, para mitigar esses ataques é necessário utilizar mecanismos de autenticação de clientes (e.g., máquinas) e controle de tráfego para reduzir o impacto de possíveis ataques. Esta parte final de prevenção de ataques de serviço já depende do ambiente onde o sistema for integrado.

Os registos não devem estar disponíveis para consulta por parte de qualquer utilizador do sistema. Apenas os administradores e entidades autorizadas têm acesso a essa informação. Como tal, o sistema deve perder o controlo sobre o conteúdo dos *logs* quando estes entram na interface do sistema, i.e., o conteúdo dos registos é cifrado com uma chave pública que não pertence ao sistema. Só o detentor da chave privada, correspondente à chave pública utilizada, pode aceder ao seu conteúdo.

1.3 Estrutura do documento

Este documento encontra-se estruturado da seguinte forma: no Capítulo 2 será feita uma descrição do trabalho relacionado e uma introdução ao conceito dos *logs*; no Capítulo 3 é apresentado um *background* das tecnologias e mecanismos utilizados neste trabalho; no Capítulo 4 é apresentado o sistema de Registo de Dados Indelével, passando pelo modelo de sistema, descrição do protocolo e dos componentes; no Capítulo 5 é detalhada a concretização do sistema; no Capítulo 6 é feita uma descrição dos testes realizados e discussão dos resultados obtidos; e por fim no Capítulo 7 apresentam-se as conclusões finais e comentado o trabalho futuro.

Capítulo 2

Trabalho relacionado

Accorsi [2] apresenta os vários intervenientes que existem num serviço de *log*. Num sistema de *logs*, os dispositivos capturam eventos e enviam-nos para os coletores que os armazenam em ficheiros *log*. Os auditores autorizados recuperam partes do ficheiro *log*, através dos coletores. As mensagens de *log* enviadas, por dispositivos para os coletores, estão em trânsito e as mensagens registadas no ficheiro *log* estão em repouso. As partes do ficheiro *log* recuperadas pelos auditores estão em processo.

Os serviços de *log* podem ser categorizados em:

- Utilização de CD-ROM ou discos WORM, imprimir diretamente em papel, entre outros, para armazenamento de *logs* [6].
- `syslog` e as extensões que lhe adicionam funcionalidades de segurança [29, 12, 33];
- Soluções baseados no esquema de Schneier-Kelsey (SK) [22].

As soluções propostas na primeira categoria são os meios tradicionais de armazenar *logs* e tratam dos *logs* em repouso; as propostas baseadas no esquema SK também tratam desses *logs*; e as soluções propostas na segunda categoria focam-se nos *logs* em trânsito e em repouso.

2.1 `syslog`

O `syslog` é uma ferramenta para *logging* que está presente em praticamente todos os sistemas Unix. O `syslog` pode ser utilizado em sistemas Windows através de software

de terceiros. Além disso, a maioria dos dispositivos de rede como *firewalls*, encaminhadores, têm capacidade de gerar mensagens `syslog`. Consequentemente, o `syslog` é o que se pode ter de mais semelhante a um standard universal de *logging*.

O `syslog` foi concebido para gerar, processar e armazenar mensagens de notificação de eventos importantes que fornecem informação aos administradores sobre os sistemas. Os componentes mais utilizados do `syslog` são o `syslogd daemon` (`syslogd`) e o `kernel log daemon` (`klogd`), representados na Figura 2.1. O `klogd` gere os *logs* do *kernel* e o `syslogd` gere os *logs* de aplicações. Os *logs* são escritos em ficheiros *log* de acordo com a configuração do `syslog`. Existem ainda algumas aplicações que produzem os seus próprios *logs* [6]. Os *daemons* (`syslogd` e o `klogd`) são executados no processo de inicialização do sistema e são ferramentas passivas, i.e., esperam por *input* de programas e de dispositivos.

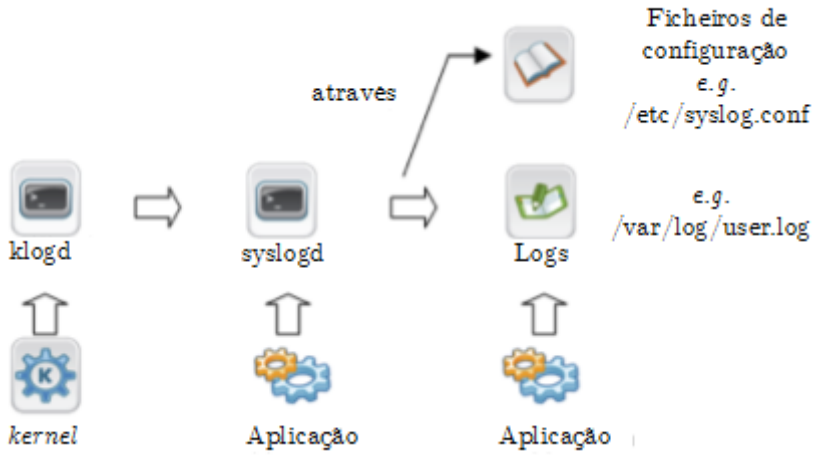

Figura 2.1: Funcionamento do `syslog` [6]

Contudo, o `syslog` tem alguns problemas de segurança. Kent et al. [15] demonstra que existem várias razões pelas quais o `syslog` não deve ser utilizado em ambientes seguros: o *sylog* foi desenvolvido numa altura em que a segurança dos *logs* não era uma preocupação. Portanto, não suporta mecanismos de segurança que preservem a confidencialidade, integridade e disponibilidade dos *logs*. O `syslog` utiliza o *User Datagram Protocol* (UDP) para transmitir informação; o UDP não dá garantias de que os registos sejam recebidos com sucesso ou na sequência correta. A maioria das implementações não faz qualquer tipo de controlo de acesso. Assim, é possível que qualquer máquina envie mensagens para um servidor `syslog` se não forem tomadas outras medidas de segurança. Os indivíduos maliciosos podem tirar partido disto para inundarem os servidores `syslog` com dados falsos. Isso pode levar a que *logs* importantes sejam perdidos ou mesmo causar

negação de serviço; a maioria das implementações não utilizam criptografia para proteger a integridade e a confidencialidade dos *logs* em trânsito. Indivíduos mal intencionados podem monitorizar as mensagens que contêm informação confidencial; os atacantes podem mesmo chegar a desempenhar ataques *man-in-the-middle*[20], como modificar ou destruir mensagens em trânsito; por fim, localmente também existem problemas, uma vez que, ao utilizar o `syslog` em computadores pessoais, os *logs* são concebidos como ficheiros normais. Um utilizador que tenha privilégios *root* poderá facilmente controlar o conteúdo dos *logs* [6].

2.2 Extensões do `syslog`

Nos últimos anos a segurança dos *logs* tem vindo a tornar-se uma preocupação, consequentemente, foram propostas várias implementações do `syslog` com maior ênfase na segurança. A maioria destas extensões foi baseada num standard proposto, o *Request for Comments* (RFC) 3195 (*Reliable Delivery for syslog*) [7]. Este RFC foi proposto especificamente para garantir:

- Confidencialidade, integridade e disponibilidade dos *logs* em repouso;
- Entrega fiável (*Transmission Control Protocol* (TCP));
- Confidencialidade de dados na transmissão (*Transport Layer Security* (TLS) ou *Secure Shell* (SSH));
- Integridade de dados na transmissão e autenticação (*Secure Hash Algorithm 1* (SHA-1)) [15].

O *syslog-sign* estende o `syslog` com assinaturas nos blocos para prevenir adulterações dos dados do *log* em trânsito [13]. Concretamente, o *syslog-sign* adiciona autenticação na origem, integridade da mensagem, resiliência a repetições, sequenciação de mensagens e deteção de mensagens em falta [14]. Contudo, esses blocos de assinatura estão fracamente acoplados às entradas dos ficheiros *log* e podem ser apagadas depois de armazenadas. Isso significa que não é dada proteção para as entradas *log* em repouso. Além disso, as entradas são transmitidas em claro [2].

Além das mensagens normais do `syslog`, o *syslog-sign* adiciona dois tipos de mensagens especiais: mensagem de bloco de assinaturas e mensagem de bloco de certificados. A mensagem de bloco de assinaturas contém as sínteses das mensagens enviadas no

passado e as assinaturas dessas sínteses. A verificação das assinaturas torna possível a autenticação das mensagens enviadas. O propósito da mensagem de bloco de certificados é suportar a gestão de chaves que utilize criptografia de chave pública.

As versões mais recentes de Unix são equipadas com o `syslog new generation` (`syslog-ng`), o sucessor do `syslog` [30]. Além de utilizar comunicação fiável, através do TCP, este também suporta *Internet Protocol versão 6* (IPv6) e troca de mensagens cifradas utilizando o protocolo TLS [2].

Em 2005 , foi proposto o SecSyslog por Forte et al. [9], uma solução inovadora para o problema da transmissão de dados em claro. Para tal foi utilizado um canal oculto, mais precisamente um canal oculto de *Domain Name System* (DNS). Por canal oculto entende-se qualquer método que permita a transmissão de informação, através de uma ou mais variáveis globais de um sistema, variáveis essas que não foram oficialmente concebidas para esse propósito. Num canal oculto de DNS são utilizados campos como o CNAME e o TXT para transmitir informação, que no total disponibilizam 330 bytes. Este canal oculto é utilizado para enviar os *logs* para o servidor DNS. Como ilustra a Figura 2.2, os clientes colocam a informação que querem registar no *log* numa atualização de DNS. O servidor SecSyslog obtém essa informação através de respostas a *queries* DNS. Nessas respostas vêm mensagens `syslog`. Esta arquitetura permite que a identidade do servidor de *log* permaneça secreta para os clientes.

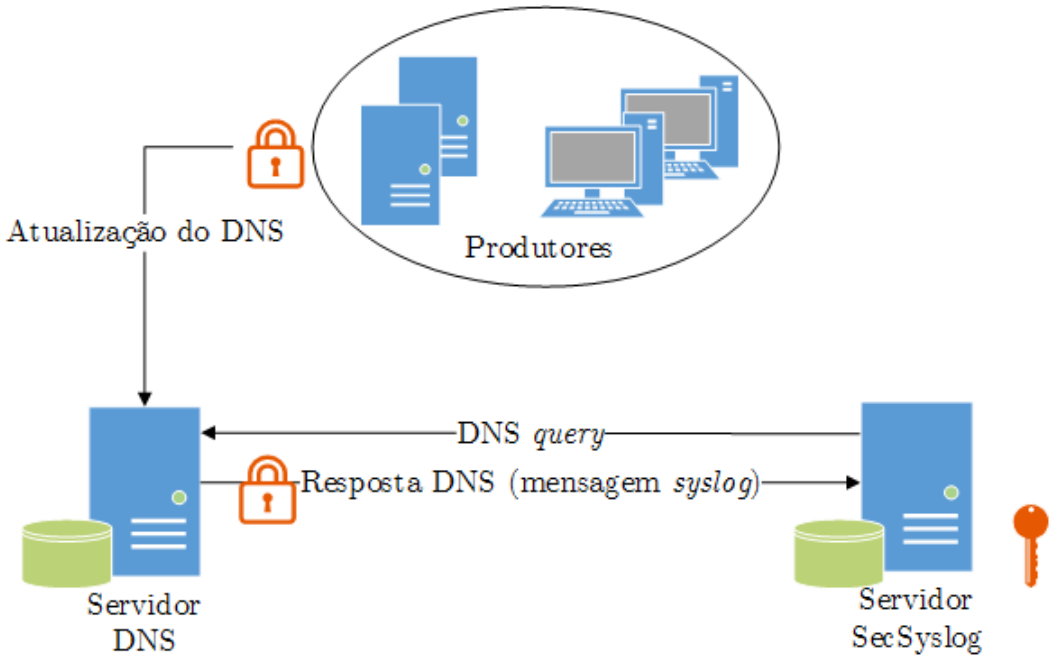

Figura 2.2: Arquitetura do SecSyslog [9]

2.3 Esquema Schneier e Kelsey

Ao contrário das extensões do `syslog`, as soluções propostas com base no esquema SK focam-se nos *logs* em repouso. Este esquema deriva de um conceito introduzido por Bellare e Yee: Segurança Futura[3]. Esta propriedade de segurança permite detetar a eliminação e modificação de *logs*, mesmo quando as chaves são obtidas pelo atacante. Para tal são utilizados *Message Authentication Code*s (MACs). Uma vez que o uso do MAC implica computação com uma chave secreta, a chave utilizada para gerar o MAC de uma mensagem deve ser posteriormente apagada, por forma a evitar que um atacante possa forjar o MAC de mensagens. Não obstante, é difícil preparar uma nova chave para cada mensagem *log* que é recebida. Este aspeto torna necessário que outra chave tenha que ser criada, derivada de uma chave base, com a condição de que seja impossível inferir a chave base a partir da chave derivada. Para obter tal propriedade, são utilizadas funções de síntese para gerar chaves. Suponhamos que K_i é a chave para computar o MAC da mensagem i e que $h()$ é a função de síntese,

$$K_1 = h(K_0), K_2 = h(K_1), ..., K_i = h(K_{i-1})$$

a chave base (K_0) deve ser guardada numa área segura. Se a chave for apagada imediatamente após a geração da próxima, a modificação e eliminação de mensagens anteriores pode ser detetada [14].

Schneir e Kelsey propõem um protocolo que utiliza a propriedade de Segurança Futura. Neste protocolo, o cliente envia a chave base (K_0) para o servidor de *log*. O cliente fica responsável por utilizar funções de síntese para gerar as próximas chaves a utilizar no MAC e a cifrar as entradas *log*.

Porém estas duas soluções têm a mesma desvantagem, o facto de o cliente e o servidor poderem modificar os *logs*. Isto acontece porque não existe uma autoridade confiável a participar no protocolo. Além disso, estas soluções têm uma vulnerabilidade em comum: o ataque "*tail-cut*" [1].

Os serviços desenvolvidos com base no esquema SK utilizam cadeias de sínteses para criar dependências entre entradas *log*. Por isso, remover uma ou mais entradas *log* da cadeia permite que os verificadores detetem a adulteração. Contudo, se o atacante remover entradas do fim do ficheiro *log* (*tail-cut*) o verificador não é capaz de detetar, a menos que saiba o tamanho da cadeia de sínteses. Recentemente surgiu outra solução, BBox, que já não sofre desta vulnerabilidade [2]. O BBox garante autenticidade, tanto dos dados em

repouso, como dos dados em trânsito, e permite pesquisa por palavras-chave nos ficheiros de *log* cifrados.

Por fim, Ma e Tsudik propuseram uma nova alternativa baseada em assinaturas acumulativas, em vez de cadeias de síntese[16]. Até ao momento, ainda não é claro se esta alternativa é vulnerável a ataques de truncamento, contudo possui a vantagem de, do ponto de vista de desempenho, ser muito mais eficiente do que os serviços que utilizam cadeias de síntese[2].

2.4 Comparação de trabalhos

Segurança para	`syslog`	`syslog-ng`	SK	Sistema proposto
logs em repouso	×	×	✓	✓
logs em trânsito	×	✓	×	✓
logs indeléveis	×	×	×	✓

Tabela 2.1: Comparação entre os trabalhos já desenvolvidos e o sistema proposto.

Na Tabela 2.1, está representada a comparação entre os trabalhos desenvolvidos anteriormente, na área dos *logs*, e o sistema proposto neste documento. A comparação é feita com base nas três propriedades de segurança que se pretende garantir.

Capítulo 3

Background

Este trabalho utiliza conceitos da área da tolerância a faltas, tolerância a faltas Bizantinas e criptografia. Para facilitar a apresentação do sistema, *Indeliable Logging*, será primeiramente apresentada uma visão global sobre alguns paradigmas e tecnologias dessas áreas.

3.1 TLS (*Transport Layer Security*)

O *Secure Socket Layer* (SSL) é um protocolo que foi criado pela Netscape para introduzir segurança em comunicações que recorrem ao protocolo *Hypertext Transfer Protocol* (HTTP). O SSL V3 é a última versão deste protocolo, entretanto padronizado com o nome de *Transport Layer Security* (TLS). O TLS é idêntico ao SSL, embora as suas diferenças sejam suficientes para criar incompatibilidade, de tal forma que há manutenção dos dois protocolos nas diversas aplicações distribuídas. Zúquete, em [40], apresenta mais detalhes sobre este protocolo e a sua finalidade.

O TLS garante uma comunicação segura cliente-servidor para transportes com ligação, nomeadamente o TCP, e é constituído por dois sub-protocolos: um que gere a criação e manutenção de sessões seguras TLS (TLS *Handshake Protocol*) e outro que gere o transporte seguro sobre um protocolo de transporte inseguro, usando para efeito os parâmetros e algoritmos criptográficos associados à sessão segura (TLS *Transport Protocol*). O transporte seguro permite efetuar compressão de dados e garante confidencialidade e controlo de integridade dos dados trocados.

Na negociação das sessões seguras ambos os negociadores, cliente e servidor, têm a possibilidade de se autenticar usando pares de chaves assimétricas e certificados X.509 da

respetiva chave pública. O TLS permite os seguintes três modelos de autenticação:

1. Nenhuma (interações anónimas) - Neste caso os interlocutores criam um canal seguro, usando uma chave partilhada negociada com o algoritmo de Diffie-Hellman[8]. A troca dos valores públicos de Diffie-Hellman não é autenticada, sendo por isso suscetível a ataques de interposição.

2. Autenticação do servidor - Neste caso os interlocutores criam um canal seguro, usando uma chave negociada entre ambos e um protocolo em que o servidor utiliza um par de chaves assimétricas e um certificado X.509 da chave pública para se autenticar. A chave pode ser negociada, utilizando o algoritmo de Diffie-Hellman ou simplesmente escolhida pelo cliente e enviada de forma secreta para o servidor, cifrada com uma chave pública do recetor.

3. Autenticação mútua cliente-servidor - Neste caso os interlocutores criam um canal seguro e usando uma chave negociada entre ambos e um protocolo em que ambos, cliente e servidor, utilizam um par de chaves assimétricas e um certificado X.509 da chave pública para se autenticar. Tal como no caso anterior, a chave pode ser negociada, utilizando o algoritmo de Diffie-Hellman, ou simplesmente escolhida pela cliente e enviada de forma secreta para o servidor, cifrada com uma chave pública do recetor.

Na maioria dos casos, o TLS é usado por aplicações que comunicavam de forma insegura usando protocolos aplicacionais sobre TCP, passando a utilizar TLS em vez de TCP - como é o caso de algumas versões mais recentes do `syslog`.

3.2 NAS (*Network-Attached Storage*)

Um dispositivo *Network-Attached Storage* (NAS) é um sistema de armazenamento para fins especiais que é acedido remotamente através de uma rede de dados, como representado na Figura 3.1. Os clientes acedem ao NAS através de uma interface de chamada remota de procedimentos como o *Network File System* (NFS)[5] para sistemas UNIX ou o *Common Internet File System* (CIFS)[18] para máquinas Windows. As chamadas remotas de procedimentos são feitas através de TCP ou UDP sobre uma rede *Internet Protocol* (IP). A unidade NAS é normalmente implementada sobre *Redundant Array of Independent Drives* (RAID) com software que implementa a interface de chamada remota

de procedimentos. Silberschatz et al. [25], apresentam o desígnio e o funcionamento dos NAS com mais detalhe.

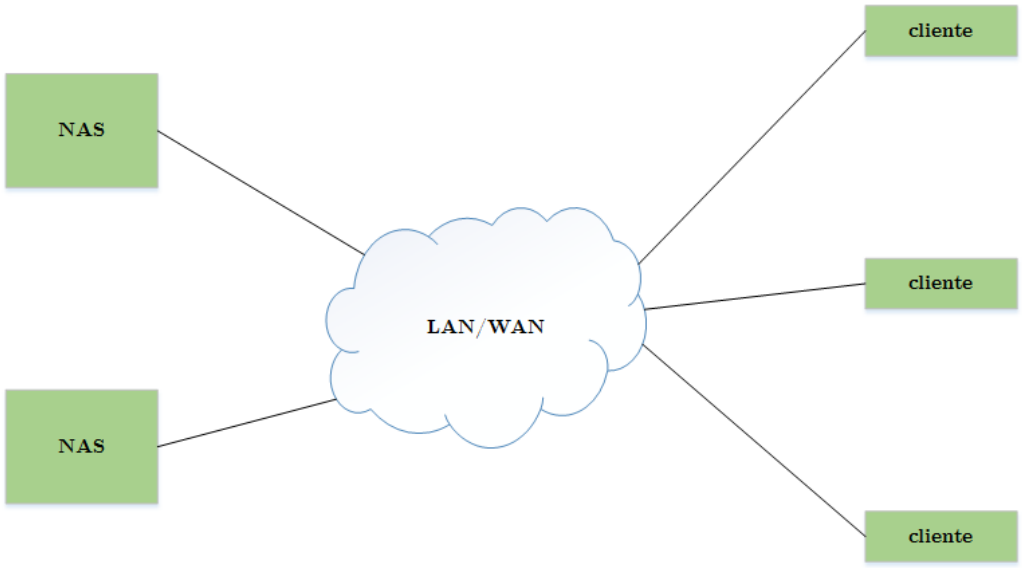

Figura 3.1: NAS.

Os computadores acedem aos discos de armazenamento de duas formas. Por um lado, podem fazê-lo através de portas através de portas de I/O (*Host-Attached Storage*), o que é comum dos sistemas pequenos. Por outro lado, podem fazê-lo através de uma máquina remota num sistema de ficheiros distribuídos, isto é referido como NAS. No âmbito deste documento, apenas será abordado o acesso a disco de armazenamento através da rede - NAS.

O NAS proporciona uma forma conveniente para todos os computadores, numa *Local Area Network* (LAN), partilharem armazenamento com a mesma facilidade de acesso e designação das soluções de armazenamento locais. No entanto, este tende a ser menos eficiente e a ter um desempenho mais baixo do que as opções de armazenamento locais porque, as operações têm de passar pela LAN antes chegarem ao armazenamento.

A tecnologia de NAS continua em evolução, sendo o *Internet Small Computer System Interface* (iSCSI)[21] um dos protocolo mais recentes. Resumidamente, este protocolo utiliza o protocolo IP para transportar o protocolo *Small Computer System Interface* (SCSI). Portanto, as redes - ao invés de cabos SCSI - podem ser utilizadas para interligação entre as máquinas e seus recursos de armazenamento.

3.3 Replicação

Entende-se por replicação o conjunto de etapas necessárias para a execução de uma série de ações em diferentes locais, com o objetivo de produzir os mesmo resultados. Veríssimo e Rodrigues dão mais detalhes do que é a replicação em [34]. A Figura 3.2 ilustra o diagrama de fluxo genérico. O mesmo pedido P_a é executado por um conjunto de participantes, num processo que envolve as seguintes fases: difusão, execução e consolidação. A difusão consiste em disseminar o pedido pelos participantes envolvidos. Dependendo do modelo gestão da replicação, o protocolo pode ou não exigir que todas as mensagens sejam entregues de forma fiável e totalmente ordenadas, de forma a reforçar o determinismo das réplicas. A execução acontece nos participantes envolvidos e, mais uma vez, depende do modelo de replicação. Por exemplo, num modelo de replicação ativa, todos os participantes executam o mesmo conjunto de pedidos na mesma ordem. Contudo, num modelo de replicação passiva, a réplica primária é a única a executar os pedidos, enquanto que as réplicas secundárias apenas criam *logs* desses pedidos e recebem atualizações de estado da réplica primária - *checkpoints*. Os resultados da execução são depois consolidados, de forma a entregar o resultado correto. Por exemplo, num modelo de faltas omissas (i.e. quando a replicação é utilizada para disponibilidade), é suficiente entregar um dos resultados, $R_a = R_a(i)$, possivelmente o primeiro a ser produzido. Contudo, se o modelo de faltas inclui faltas de valor, então é necessária uma votação por maioria entre os resultados das réplicas, i.e., $R_a = votação(R_a(1), ..., R_a(n))$. Esta forma de consolidação requer que os participantes executem um algoritmo, após o qual o resultado é retornado. Alternativamente, a consolidação pode ser feita no destino, i.e., todos os resultados $R_a(i)$ são entregues e o destinatário determina o resultado final. Apesar de menos frequente, é eficiente especialmente em computações replicadas recursivas ou em cascata, como é o caso do sistema de *logs* apresentado neste documento.

3.4 Virtualização

A ideia fundamental por detrás das máquinas virtuais é abstrair o hardware de um computador (a *Central Processing Unit* (CPU), a memória, os discos, as interfaces de rede e por aí adiante) por vários ambientes de execução, criando, desta forma, a ilusão de que cada ambiente de execução está a executar no seu próprio computador. Em [25], Silberschatz et al., fazem uma apresentação simples e intuitiva sobre a virtualização, sem abordar detalhes menos relevantes para este trabalho.

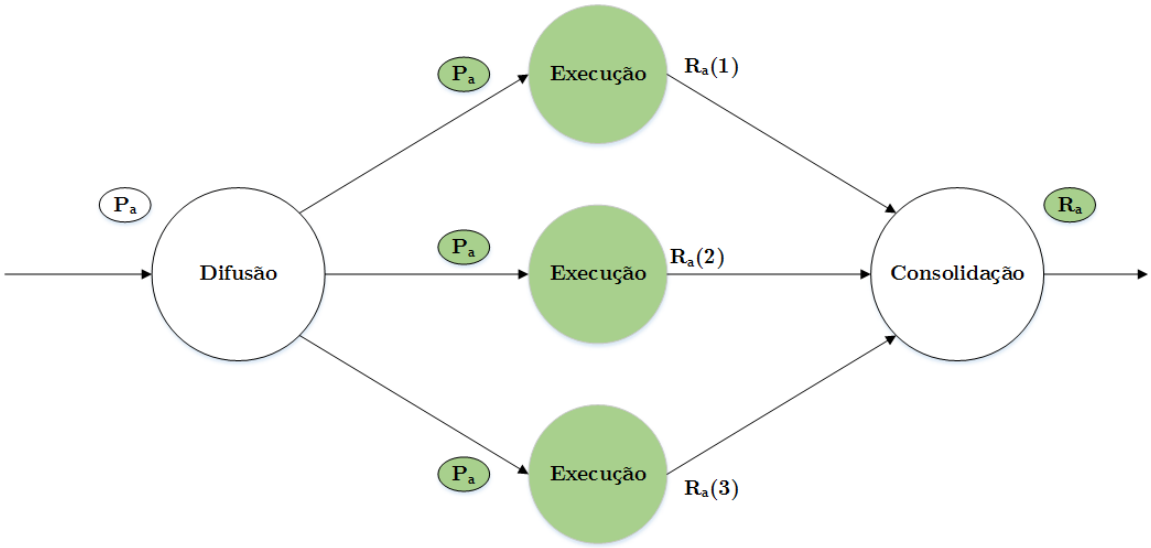

Figura 3.2: Diagrama de fluxo das atividades da replicação.

Através do escalonamento da CPU e de técnicas de memória virtual, um sistema operativo "nativo" (*hypervisor*) pode criar a ilusão de que um processo tem o seu próprio processador e a sua própria memória (virtual). A máquina virtual fornece uma interface que é idêntica (ou não no caso de emulações) ao hardware subjacente. A cada processo "alojado" é fornecida uma representação (virtual) do computador subjacente. Normalmente, o processo "alojado" é de facto um sistema operativo, e assim obtém-se um ambiente em que várias máquinas virtuais acedem a um único (*hypervisor*), que é uma camada de software que abstrai o acesso aos recursos da máquina real, garantindo assim o isolamento das máquinas virtuais.

Existem várias razões para a criação de máquinas virtuais. A maioria delas estão fundamentalmente ligadas à possibilidade de partilhar o mesmo hardware enquanto se executam, de forma concorrente, vários ambientes de execução diferentes.

Uma vantagem importante é que o sistema nativo é protegido das máquinas virtuais, tal como as máquinas virtuais são protegidas umas das outras. Um vírus num sistema operativo "alojado" pode danificar esse sistema operativo, mas é improvável que afete os outros sistemas "alojados" ou o sistema "nativo". Como cada máquina virtual é completamente isolada das outras, não existem problemas de proteção.

Uma vez as máquinas protegidas entre si, não existe partilha direta de recursos. Foram implementadas duas formas de possibilitar a partilha entre máquinas virtuais. A primeira realiza-se através de um sistema de ficheiros partilhado e, como tal, na partilha de ficheiros. A segunda baseia-se na utilização de uma rede privada entre as máquinas virtuais,

em que cada uma delas pode enviar informação pela rede de comunicação virtual.

Exemplos de software de virtualização através de máquinas virtuais são o Xen[39] e o VMware[36].

3.5 NIDS (*Network Intrusion Detection System*)

Um detetor de intrusões é uma tecnologia de segurança que tenta identificar e isolar intrusões em sistemas computacionais. Existem dois grandes tipos de detetores de intrusões (DI): os instalados nas redes (*Network Intrusion Detection System* (NIDS)) e os instalados localmente nas máquinas (*Host-based Intrusion Detection System* (HIDS)). Os HIDS monitorizam e analisam internamente um sistema computacional e, em alguns casos, os pacotes de rede que passam nas suas interfaces de rede. Os NIDS tentam descobrir acessos não autorizados a uma rede de computadores através análise do tráfego nessa rede para detetar sinais de atividade maliciosa. Neste caso, o interesse está apenas virado para os NIDS. Ptacek e Newsham, em [32], apresentam mais detalhes e formas de utilização de DIs.

Existem vários tipos de intrusão de acordo com os diferentes sistemas de DI. Por um lado, um sistema que tenta detetar ataques direcionados a um servidor Web pode ter em conta apenas pedidos HTTP maliciosos, por outro lado, um sistema que monitorize protocolos de encaminhamento dinâmico pode apenas considerar ataques de *spoofing* ao protocolo *Routing Information Protocol* (RIP). Contudo, a definição de intrusão é abrangente a todos os sistemas de DI: uso não autorizado ou indevido de um sistema computacional.

A deteção de intrusões é um componente importante de um sistema de segurança e complementa outras tecnologias de segurança. Ao fornecer informação ao administrador, o sistema de DI permite não só detetar ataques explicitamente abordados por outros sistemas de segurança, como por exemplo *firewalls*, mas também procura notificar ataques nunca antes vistos por outros componentes de segurança. Os sistemas de DI também fornecem informação forense que permite às organizações descobrir a potencial origem de um ataque.

Existem duas técnicas de deteção utilizadas pelos sistemas de DI: baseadas em anomalias estatísticas ou baseadas em assinaturas. Um sistema DI baseado em anomalias estatísticas determina a atividade normal da rede (e.g., a quantidade de largura de banda utilizada, quais os protocolos utilizados, os portos abertos e que dispositivos costumam comunicar, etc.). Quando o sistema de DI deteta tráfego que é fora do normal alerta

o administrador [38]. Por outro lado, um sistema de DI baseado em assinaturas monitoriza os pacotes de uma rede e compara com padrões de ataques pré-configurados e pré-determinados, conhecidos como assinaturas.

Um exemplo de um NIDS é o `Snort`[40, 26], o escolhido para o sistema de Registo de Dados Indelével. O `Snort` é uma ferramenta eficiente e flexível de inspeção de rede que possui um motor simples de monitorização de tráfego de rede, que é capaz de detetar uma grande variedade de ataques em redes de pequena e média dimensão; as redes de grande dimensão necessitam de um conjunto de regras demasiado grande.

Em termos arquiteturais, o `Snort` é formado por uma série de camadas funcionais que facilitam a divisão de tarefas, desde a captura de informação de rede, até à sua análise de alto nível para determinação de ataques em curso em tempo real.

Ao nível mais baixo, utiliza uma biblioteca genérica de captura de pacotes IP, a `libpcap`. Acima da biblioteca de captura, são utilizados módulos de descodificação de pacotes, os quais fornecem uma informação mais estruturada sobre os blocos de octetos que são os pacotes IP (separação e identificação de cabeçalhos, identificação de protocolos, etc.).

Acima destes módulos, é utilizado um conjunto flexível de pré-processadores que têm como função fazer uma análise prévia e uma normalização da informação capturada para tornar efetiva uma deteção correta de padrões de ataque. Os pré-processadores incluem módulos de reconstrução de informação (desfragmentação de pacotes IP, reconstrução e seguimento de fluxos TCP), módulos de deteção de atividades de procura de portos abertos, módulos de deteção de tráfego anormal (que podem servir para alimentar técnicas de deteção de intrusões baseadas em comportamento) e módulos de normalização de protocolos aplicacionais (HTTP, `telnet`, etc.). Os módulos de reconstrução e normalização são críticos para todas as atividades subsequentes de deteção de padrões textuais em trocas de dados.

O motor de análise do `Snort` baseia-se em regras definidas numa linguagem própria. Cada regra especifica um conjunto de padrões característicos de um pacote pertencente a um ataque conhecido. Tipicamente, esses padrões são o sentido da comunicação, os endereços IP de origem e destino, o protocolo de transporte, o protocolo aplicacional (inferido a partir de portos-padrão usados pelos mesmos ou por portos dinâmicos trocados em protocolos aplicacionais acompanhados) e padrões presentes nos dados aplicacionais trocados. Mas podem ser utilizados outros padrões, por exemplo, combinações anormais de opções em cabeçalhos TCP.

A lógica do motor `Snort` consiste em percorrer todas as regras ativas até encontrar uma que se adeque aos dados em causa, sendo essa regra usada para tomar alguma decisão acerca dos mesmos. Essa decisão pode incluir: aceitar os dados como normais, registar apenas a sua ocorrência ou acionar um alerta. Para acelerar a pesquisa das regras, estas são agrupadas numa tabela de dispersão, onde cada lista está afeta a uma entrada constituída por endereços IP e portos de origem e destino. Desta forma, muitas regras não aplicáveis a um dado pacote IP, podem ser facilmente evitadas na análise do mesmo.

3.6 `iptables`

Os *kernels* Unix atuais possuem a capacidade de análise de pacotes que chegam, partem ou atravessam a máquina através de um componente chamado `iptables`.

O `iptables` é um módulo do *kernel* do Linux que recebe todos os pacotes que destinados são à maquina e que vão ser enviados por esta. O destino desses pacotes depende das regras programadas no módulo. Essas regras podem ser introduzidas pelo administrador do sistema após o início de atividade do módulo. Zúquete, em [40], dá mais detalhes sobre o funcionamento do `iptables`.

A funcionalidade intrínseca do `iptables` permite realizar apenas uma *firewall* do tipo filtro de pacotes. No entanto, o `iptables`, através de redirecionamento de pacotes, permite que os fluxos de informação sejam redirecionados para quaisquer aplicações locais. O *kernel* fornece também os mecanismos-base para a utilização de outros tipos de *firewall*, nomeadamente filtros de circuitos ou aplicacionais. Contudo, a funcionalidade de tais filtros é completamente independente do `iptables`.

O *kernel* utiliza o conceito de cadeias para analisar pacotes. Uma cadeia é uma sequência de regras e cada regra possui zero ou mais condições de aplicabilidade e uma decisão. Todos os pacotes que passam pelo *kernel* são verificados no mínimo por uma cadeia, cujas regras são testadas até se encontrar uma passível de ser aplicada ao pacote. Da aplicação dessa regra deriva uma decisão acerca do futuro do pacote. Caso nenhuma regra seja aplicável, o pacote segue viagem.

O *kernel* possui cinco cadeias-padrão, mas podem ser criadas outras para reutilizar regras. As cadeias-padrão são *INPUT*, *OUTPUT*, *FORWARD*, *PREROUTING* e *POSTROUTING*. A *INPUT* aplica-se a pacotes recebidos pela máquina e que lhe são direcionados; a *OUTPUT* aplica-se a pacotes enviados pela máquina; a *FORWARD* aplica-se a pacotes recebidos pela máquina mas que não lhe são direcionados, ou seja, que passam

em trânsito com outro destino; a *PREROUTING* aplica-se a todos os pacotes recebidos pela máquina e a *POSTROUTING* a todos os pacotes transmitidos pela máquina.

O `iptables` utiliza tabelas para subdividir a aplicação de regras em cada cadeia. Estas tabelas servem para agrupar modelos de operação e dependem do modo como o `iptables` foi criado e instalado e de outros módulos instalados no *`kernel`* Linux.

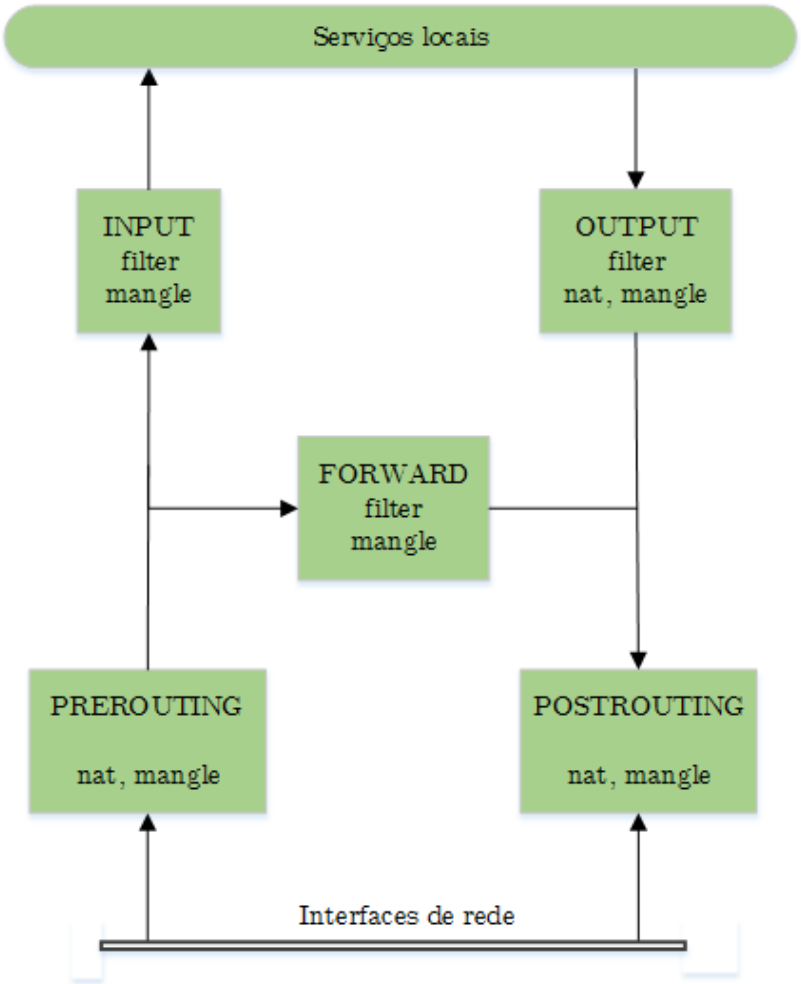

Figura 3.3: Fluxo de encaminhamento de pacotes através das cadeias e tabelas do `iptables`.

Existem três tabelas-base: `filter`, `nat` e `mangle`. A tabela `filter` existe sempre por omissão e serve para filtrar pacotes, ou seja, para decidir apenas sobre a aceitação ou rejeição. A tabela `nat` serve para detetar e atuar em situações em que seja necessário fazer *`Network Address Translation`* (NAT). A tabela `nat` serve para efetuar diversos tipos de alterações nos pacotes, como o IP de destino, porto, etc..

A afetação de tabelas a cadeias está ilustrada na Figura 3.3. Nem todas as tabelas existem em todas as cadeias porque tal não é necessário. Por exemplo, não existe tabela

`filter` nas cadeias *PREROUTING* e *POSTROUTING* e não existe `nat` nas cadeias *INPUT* e *OUTPUT*.

A decisão expressa por cada regra é uma decisão-padrão ou o nome de outra cadeia. Neste último caso, existe uma delegação, porque a decisão deverá ser tomada pelas regras dessa cadeia. Existem quatro decisões-padrão básicas e diversas extensões.

As decisões-padrão são *ACCEPT*, *DROP*, *QUEUE* e *RETURN*. A primeira indica que o pacote deve ser aceite, a segunda que ele deve ser descartado, a terceira que o pacote deve ser enviado para uma fila de espera afeta a uma aplicação local e a quarta indica que a cadeia atual deve ser abandonada e retomada a análise de regras na regra seguinte da cadeia anterior.

As decisões-extensões são diversas, entre elas estão as padrões *LOG* (para registar ocorrência do pacote em ficheiros de registo), *MARK* (para mudar uma marcação local do pacote), *REJECT* (para rejeitar um pacote com uma mensagem de erro para o emissor), *TOS* (para laterar o campo *Time of Service* do pacote), *SNAT* (para realizar *Source nat*), *DNAT* (para realizar *Destination nat*), *MASQUERADE* (para realizar *masquerading*) e *REDIRECT* (para redirecionar o pacote para um porto local).

É possível associar, em seguida, uma decisão por omissão às cadeias-padrão, caso nenhuma anterior seja aplicável. Esta decisão é designada por política, não podendo ser nenhuma outra cadeia senão uma decisão-padrão ou uma extensão.

As vantagens e desvantagens do `iptables` encontram-se sumariadas na Tabela 3.1.

3.7 TPM (Trusted Platform Module)

A *Trusted Platform Module* (TPM) é um *chip* instalado na *motherboard* de um computador. Este *chip* é um componente de *hardware* que precisa de suporte em *software* para ser utilizável. Miguel Correia et al.[17] dá mais pormenores sobre este módulo. O futuro desta plataforma ainda não é claro e as suas aplicações ainda são temas de investigação. Contudo, duas funções básicas que estão na base das utilizações do TPM são:

- Armazenamento seguro de chaves criptográficas usadas para operações particularmente críticas, como identificação de um computador ou armazenamento seguro de chaves criptográficas de mais curta duração;
- Verificação de integridade do sistema, i.e., verificar se o *software* em execução num

Vantagens

1. Produto comparável em eficiência às demais *firewalls* comerciais
2. Confiável e escalável
3. Estável
4. É apenas o *kernel* de uma arquitetura mais complexa e extensível
5. Desenvolvido, testado e melhorado por uma grande comunidade de utilizadores
6. Económico, em termos de recursos computacionais necessários

Desvantagens

1. É necessário entender a interação entre o *kernel* Linux, o `iptables` e diversos outros módulos que interagem com os dois anteriores.

Tabela 3.1: Vantagens e desvantagens do `iptables`.

determinado computador corresponde àquilo que se pretende.

A Figura 3.7 apresenta uma representação da arquitetura de uma máquina com um TPM e o software que permite utilizar este componente. Este *software* tem dois componentes básicos: *driver* e a biblioteca dos *Trusted Support Services* (TSS). Uma aplicação que precise de interagir com o TPM chama funções da biblioteca que comunicam com o *driver* e este com o TPM.

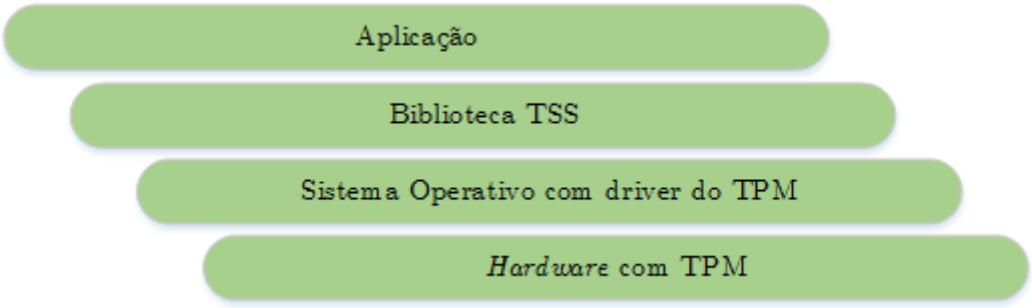

Figura 3.4: Arquitetura de *hardware* e *software* com o TPM.

3.8 TCB (Trusted Computing Base)

Uma *Trusted Computing Base* (TCB) de um sistema computacional é o conjunto de todo o *hardware*, *firmware*, e/ou componentes *software* que são cruciais para a sua segurança, no sentido em que, os erros ou vulnerabilidades que ocorrem dentro da TCB, podem comprometer as propriedades de segurança do sistema inteiro. Por outro lado, as partes do sistema computacional que ficam fora da TCB, não devem de ser capazes obter mais privilégios do que os que lhes são concedidos pela política de segurança.

O desenho e implementação da TCB de um sistema, é fundamental para a sua segurança. Os sistemas operativos modernos esforçam-se para reduzir o tamanho das TCBs, para que uma análise exaustiva do seu código (por meio de software de auditoria manual ou auxiliada por computador) seja possível de realizar.

Capítulo 4

Registo de Dados Indelével

Neste capítulo será apresentado o modelo de sistema, partindo do modelo te faltas passando pelos pressupostos sobre a rede e protocolos. Em seguida é apresentado o sistema e o comportamento do sistema. Por fim é feita uma caracterização dos componentes do sistema.

O sistema é composto por quatro partes: *proxies*, a interface do sistema; servidores réplica, onde está a complexidade aplicacional; agregador, uma TCB que faz agregação de resultados; e um NAS, onde serão guardados os registos em repouso.

As *proxies* são a interface do sistema para o exterior e recebem os *logs* dos produtores. De seguida dissemina-os pelos servidores réplica. A *proxy* também tem um um filtro de pacotes. Os servidores réplica estarão a executar o código de um servidor `syslog`. Quando terminar de tratar o *log* enviam-no para o agregador. O agregador, uma TCB, recebe as mensagens dos servidores réplica e agrega-os passando apenas uma cópia dos registos para o NAS. O NAS tem um disco WORM.

4.1 Modelo de sistema

O sistema tem um modelo de faltas híbrido[35], i.e., diferentes componentes terão diferentes assunções. Contudo, não são feitos pressupostos sobre os tempos de execução ou de troca de mensagens, i.e., nenhum componente depende de um relógio ou de informação temporal para desempenhar a sua função, assim o sistema é assíncrono. Deste modo, este apresenta, à partida, menos vulnerabilidades do que um sistema síncrono. Não dependendo do fator tempo, ataques como atrasar mensagens, adulterar campos de tempo,

atrasar relógios de servidores, não terão qualquer efeito. Esta despreocupação com o tempo faz com que seja mais fácil tornar este tipo de sistemas resistente a ataques.

Todos os servidores réplica arrancam no mesmo estado, ou seja, têm o mesmo estado inicial. Os servidores e os clientes conhecem as chaves necessárias para poderem estabelecer canais seguros entre si. Os servidores réplica e os produtores de registos que são corretos não se desviam do comportamento definido, i.e., adotam o comportamento de um autómato. Os servidores não corretos podem desviar-se arbitrariamente do comportamento definido, podendo assumir um comportamento malicioso. Esta classe irrestrita de faltas é conhecida como Bizantina. O sistema é composto por n servidores onde, $n = 2f + 1$ com canais autenticados, onde f é o número de servidores que podem ser comprometidos.

Um indivíduo com intenções maliciosas pode conseguir aceder à rede e modificar mensagens, para não se perder a integridade e confidencialidade dos dados é utilizado o TLS[31], como proteção contra essas situações. Contudo, não se garante nada contra ataques de indivíduos com acesso físico às máquinas.

Para as *proxies* assume-se um modelo de faltas por paragem, i.e., as *proxies* apenas falham quando deixam de executar o seu código. Pressupõe-se que as *proxies* são seguras. Esta propriedade pode ser assegurada com grande cobertura por duas razões: o software que utilizam é muito pequeno (cerca de 200 linhas de código); e as *proxies* só terão o porto 514 aberto para a comunicação. Isto faz com que as *proxies* sejam suficientemente simples para utilizar as técnicas típicas de proteção como: minimizar os componentes do sistema operativo; fechar todos os serviços de rede; corrigir todas as vulnerabilidades conhecidas e assegurar que todos os inputs são validados. O sistema tem, no mínimo, duas *proxies* e o número máximo não é limitado.

Para as TCBs assume-se um modelo de faltas por paragem, i.e., apenas falham quando deixam de executar o seu código. As TCBs são simples e de fácil verificação, portanto são seguras e confiáveis.

Além dos servidores réplica e da *proxies*, o sistema tem ainda um TPM em cada *proxy*, com as chaves necessárias previamente instaladas e com um serviço denominado Gerador de Identificadores Únicos (GIU). O GIU coloca identificadores únicos em cada mensagem que cada *proxy* recebe. Os TPMs realizam operações criptográficas nos pacotes que recebem da rede, isto é, o conteúdo dos eventos de *log* são cifrados.

O NAS terá poderá ter falhas de disco e, como tal, deve ser utilizado um esquema de RAID 5 para ser tolerante a faltas.

Os algoritmos criptográficos são assumidos como seguros e invioláveis, significando isto que um indivíduo malicioso não consegue quebrar uma cifra, encontrar colisões para uma síntese ou forjar assinaturas.

Por fim, pressupõe-se que o *hypervisor* utilizado é seguro e inviolável, ou seja, não é possível que um indivíduo malicioso aceda ao hardware da máquina física através da máquina virtual que comprometeu. Apesar da segurança do *hypervisor* ser um tema relativamente recente, já existem alguns avanços nesta área [28, 37].

4.2 Registo de Dados Indelével- Como tornar o `syslog` seguro

Nesta secção começar-se-á pela configuração mais vulgar do `syslog` e percorrer-se-ão todas as alterações necessárias, até se alcançar a configuração que garante as propriedades pretendidas.

4.2.1 `syslog` local

Figura 4.1: Computador utiliza `syslog` localmente.

Na Figura 4.1 está representada a configuração da maioria das máquinas que utilizam `syslog`. Nesta configuração, o `syslog` escreve os *logs* da máquina em questão num repositório de registos local. O problema desta configuração é evidente: numa situação em que a máquina seja comprometida, o acesso aos *logs*, por parte do atacante, é imediato.

Se os registos de cada máquina forem armazenados localmente, torna-se difícil de garantir a sua segurança. Por exemplo, numa empresa cada utilizador instala do seu computador os programas que bem entende. Um desses programas pode ter código malicioso para comprometer o computador e aceder aos *logs*.

A solução para este problema dá origem à próxima configuração.

4.2.2 `syslog` com servidor remoto

Figura 4.2: Computador utiliza servidor `syslog` remoto.

Na Figura 4.2 está representada a configuração das máquinas de um centro de dados onde se utiliza um servidor `syslog`. Nesta configuração, as máquinas enviam os *logs* para o servidor `syslog` remoto, podendo ou não, manter cópias locais desses *logs*. A questão que se coloca com esta configuração é idêntica à anterior, sendo uma das principais diferenças, o facto de o local do problema ter sido movido para o servidor remoto. À alteração da sua localização, acresce o aumento da sua magnitude. Na configuração anterior, se a máquina fosse comprometida os *logs* locais deixariam de estar seguros. Neste caso, se o servidor for comprometido, os *logs* de todas as máquinas que utilizam o servidor são comprometidos (no caso dos produtores não terem cópia local). Novamente, a solução deste problema dá origem à configuração que se segue.

4.2.3 `syslog` com servidor remoto replicado

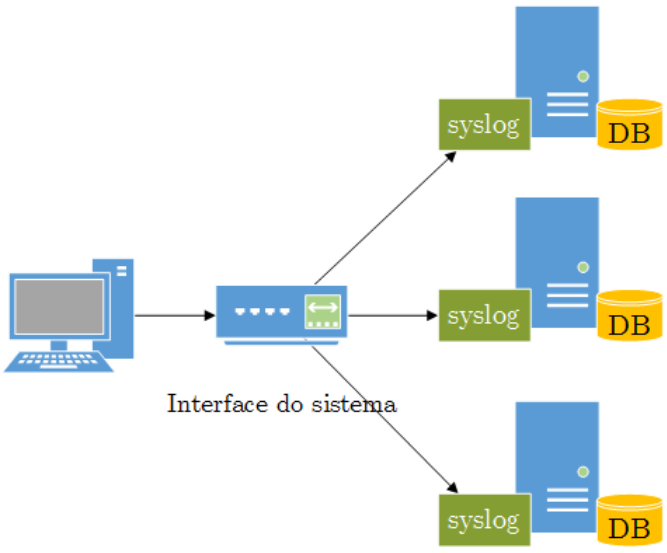

Figura 4.3: Computador utiliza vários servidores `syslog` replicados através de uma *proxy*.

Na Figura 4.3 está representada a configuração das máquinas que utilizam um servidor `syslog` remoto replicado. Para enviarem *logs* para os servidores, estas utilizam uma interface única (*proxy*), que posteriormente distribui esse registo pelos servidores. Nesta configuração, tem-se n servidores, onde $n = 2f + 1$, podendo consequentemente existir até f servidores comprometidos, onde $f = (n - 1)/2$. Desta forma para recuperar os *logs* tem-se de fazer algum tipo de votação para descartar os registos de um possível servidor comprometido. No exemplo da Figura 4.3 tem-se $f = 1$, ou seja, um servidor comprometido e dois corretos. Com esses dois servidores corretos asseguramos que os *logs* corretos são escolhidos no final de todas as votações. Se, com a mesma configuração da Figura 4.3 tivermos, por exemplo $f = 2$, já não é possível recuperar uma cópia dos *logs* corretos. Como os dois servidores comprometidos estão em maioria, estes podem combinar mensagens forjadas e fazer o servidor correto passar por impostor.

Existem dois problemas nesta configuração: a posterior recuperação dos *logs* e a perda de confidencialidade dos *logs* num servidor comprometido. Por forma a recuperar os *logs*, seria necessário existir, no final, algum tipo de agregação, de modo a que, de um total de três, se obtivesse apenas uma cópia dos *logs*. Se se considerar uma elevada quantidade de *logs*, este problema pode consumir um período alongado de tempo. No que respeita a evitar a perda de confidencialidade dos *logs*, é necessário que estes sejam cifrados quando entram no sistema. O modo mais simples de o fazer é através de um TPM criptográfico na *proxy*, ou seja, através da utilização de um módulo criptográfico, a *proxy* cifra o conteúdo de todos os registos antes de serem enviados para os servidores. A chave utilizada para cifrar os *logs* não pertence ao sistema, portanto, é da responsabilidade da entidade detentora da chave (administrador) garantir a sua segurança e, mais tarde, decifrar os *logs*.

Para resolver a questão da recuperação dos *logs*, é necessária a configuração seguidamente apresentada.

4.2.4 `syslog` com servidor remoto replicado e sistema de ficheiros único

Na Figura 4.4 está representada a configuração das máquinas que utilizam um servidor de *logs* remoto e replicado com um sistema de ficheiros único entre as réplicas. A *proxy* utiliza um TPM criptográfico para cifrar o conteúdo de todos os *logs* com uma chave de uma entidade exterior ao sistema. Só essa entidade tem acesso ao conteúdo dos *logs*.

Nesta configuração, para cada registo, os servidores chegam a acordo sobre o que se

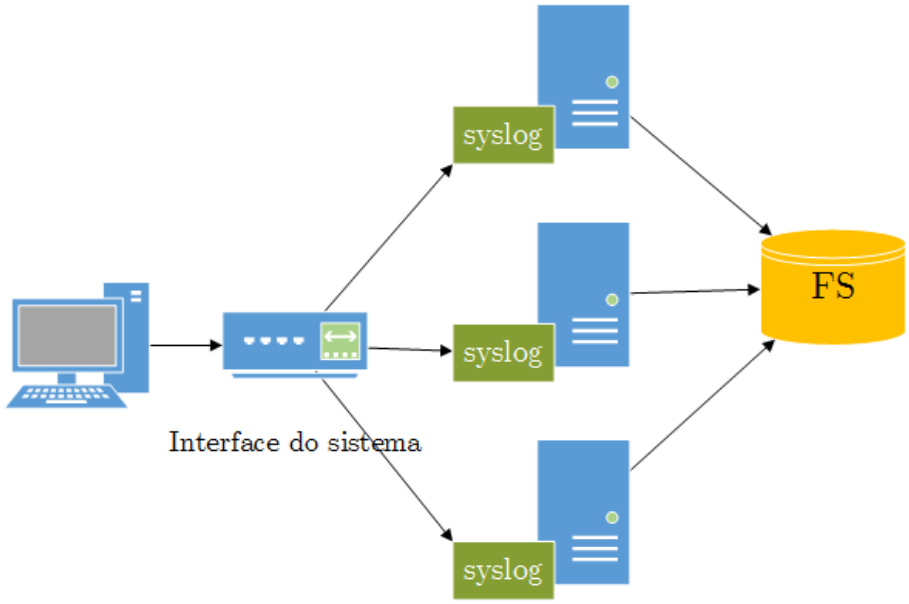

Figura 4.4: Através de uma *proxy*, o computador utiliza vários servidores `syslog` replicados, com um sistema de ficheiros únicos.

registará na entrada *log* correspondente no sistema de ficheiros e qual deles o fará. Outra alternativa consiste em todos os servidores escreverem simultaneamente no sistema de ficheiros e, deste modo, existirem entradas replicadas. Para que esta última solução seja viável, seria necessário que existisse um agente que permitisse filtrar entradas repetidas, eliminando possíveis ataques - entradas *log* de servidores comprometidos.

Todavia, não se pretende que tal ocorra, uma vez que para futuro processamento dos registos, as entradas replicadas e até maliciosas, teriam de ser removidas, o que envolve recursos computacionais. A configuração seguidamente apresentada resolve este obstáculo.

Na configuração ilustrada na Figura 4.5 introduz-se um elemento agregador (TCB) como mediador de acesso ao sistema de ficheiros (NAS). Este elemento agregador trata de garantir que, no final, o NAS só contém uma cópia de cada entrada *log*, ou seja, para cada entrada *log* que os servidores tentam escrever, ao invés de serem escritas 3 entradas iguais, apenas se escreve uma. Para tal, este elemento coleta $f + 1$ mensagens iguais, por parte dos servidores, para que registe essa entrada *log* no NAS. As *proxies* utilizam o GIU para facilitar a tarefa do agregador de comparar mensagens. Desta forma, é possível assegurar que é efetuado unicamente o registo dos pedidos corretos, sendo que um servidor comprometido perde a capacidade de colocar informação indesejável no NAS.

Para garantir que os registos escritos no NAS não são corrompidos é utilizado um disco WORM no mesmo.

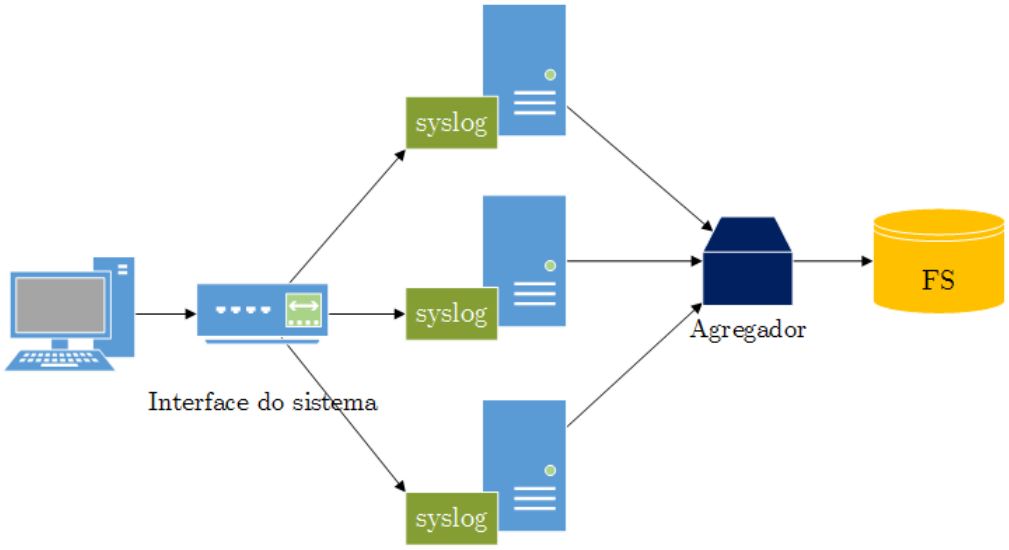

Figura 4.5: Através de uma *proxy*, o computador utiliza vários servidores `syslog` replicados, com um sistema de ficheiros únicos gerido por uma TCB.

4.2.5 `syslog` com servidor remoto replicado, sistema de ficheiros único e detetor de intrusões (recuperação pró-ativa e reativa [27])

Na Figura 4.6 está representada uma configuração onde as máquinas utilizam um servidor de *logs* remoto e replicado com um sistema de ficheiros único partilhado pelas réplicas. Como mediador entre as réplicas e o sistema de ficheiros está um agregador. A monitorizar a rede está um detetor de intrusões de rede. Com a introdução do detetor de intrusões pretende-se: tornar o sistema resiliente a longo-termo, aumentar o poder de resposta do sistema a ataques e diminuir a necessidade de monitorização humana para deteção de ataques pois, tendo uma automatização da deteção de intrusões a resposta é mais rápida e eficiente, e não requer recursos humanos para identificar o problema.

O detetor de intrusões é um agente passivo na rede, de forma a que não seja alvo de ataques. O seu papel é escutar todo o tráfego que passa pelas redes do sistema e identificar mensagens que indiquem o comprometimento de uma máquina.

Com o detetor de intrusões instalado nas redes do sistema é possível detetar mensagens forjadas que não fazem parte do sistema. Essas mensagens podem ser *logs* falsos, ataques a outras réplicas, fugas de informação sobre o sistema para o exterior, etc.. Quando o detetor identifica uma réplica fora do seu comportamento, pode acionar mecanismos de recuperação, como reiniciar a máquina e arrancar com um sistema operativo diferente do anterior.

A recuperação reativa é uma técnica que deve ser utilizada com cautela, pois um atacante puder levar o sistema a causar um ataque a si próprio. Contudo, o sistema proposto é tão simples na forma como reage, que não é possível que cause uma negação de serviço a si mesmo. Uma máquina só é alvo de recuperação quando se desvia declaradamente do comportamento expectável, como enviar mensagens para destinos que não estão previstos. Teremos n máquinas, onde $n = 2f + 1 + k$, em que f são maliciosas e k estão em processo de reiniciar, desta forma continuamos com $n - f - k$ máquinas, as suficientes para que o sistema possa cumprir o seu serviço.

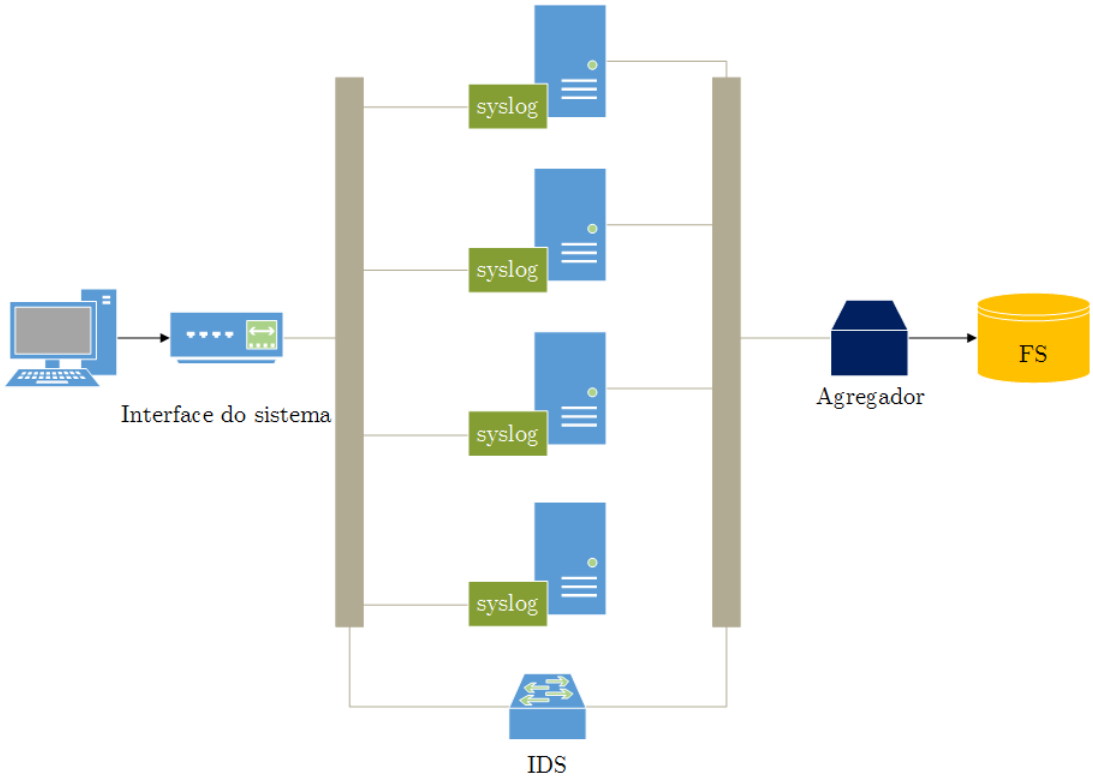

Figura 4.6: Configuração final do sistema.

A recuperação pró-ativa consiste em reiniciar as máquinas sequencialmente de forma a eliminar qualquer tipo de comprometimento não identificado pelo detetor de intrusões. Contudo, o número de máquinas a recuperar, quer por recuperação pró-ativa, quer por recuperação reativa, não deve exceder k máquinas. O sistema é assíncrono, logo não existe uma noção de tempo para facilitar a recuperação pró-ativa. Apesar disso, o comportamento das máquinas em relação à quantidade de *logs* que fazem é relativamente estável para cada ambiente. Por exemplo, um centro de dados de bases de informação de clientes de um supermercado, vai produzir mais registos do que o computador da secretária de uma empresa mas, tanto o centro de dados como o computador da secretária

produzem, sensivelmente, o mesmo volume de tráfego de *logs* por dia. Assim pode-se definir um volume de *logs* para o qual os servidores iniciariam a recuperação pró-ativa em *Round Robin*. Esse volume de *logs* depende do ambiente em questão.

No exemplo da Figura 4.6, tem-se $n = 4$, com $f = 1$ e $k = 1$. Com esta configuração uma das máquinas pode estar comprometida - máquina A - e outra a reiniciar - máquina B -, todavia o sistema continua a disponibilizar o serviço a que se propõe. Quando a máquina B terminar o processo e o detetor de intrusões detetar que máquina A está comprometida irá avançar com o reinicio dessa máquina.

4.3 Arquitetura

Na Figura 4.7 está representada a arquitetura de alto nível do sistema e a separação entre a rede das *proxies* e dos servidores, e a rede dos servidores e do agregador. Para tal, foram utilizadas duas interfaces de rede, uma para cada uma das redes. Como o NIDS monitoriza as duas redes, também tem duas interfaces de rede, uma para a Rede 2 e outra para a Rede 3.

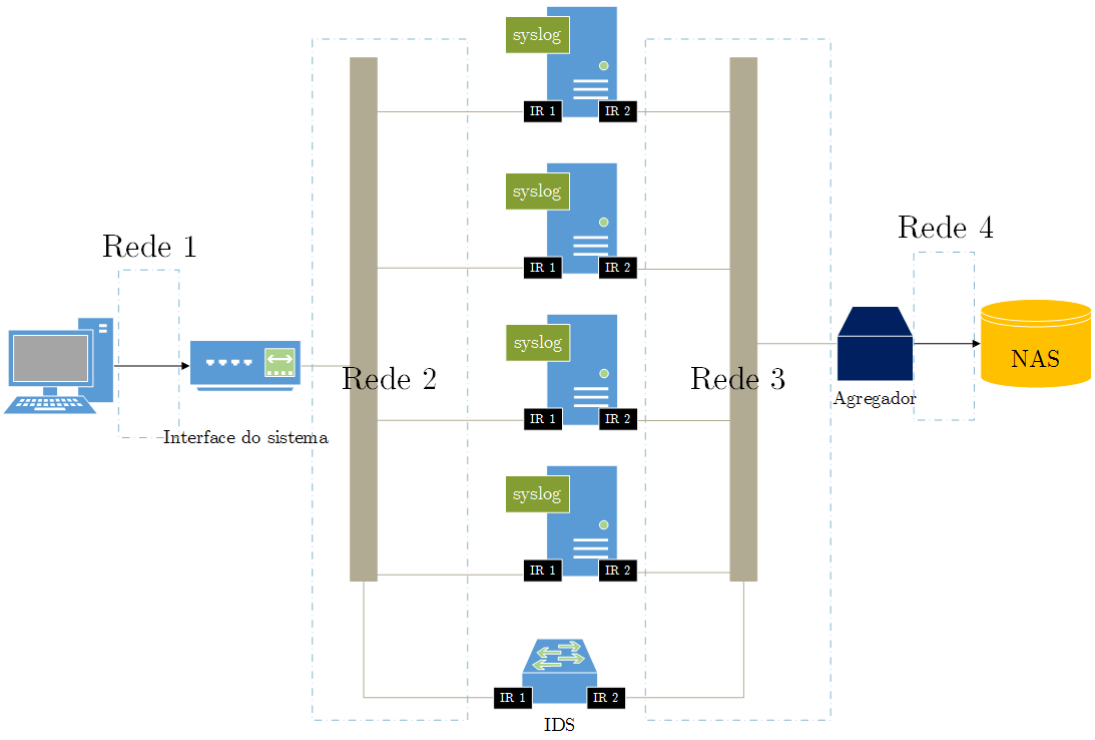

Figura 4.7: Redes que compõem o sistema de *logs*. Isolamento dos componentes, por meio de várias redes.

O isolamento é importante para controlar a capacidade de comunicação dos servido-

res. Como a informação segue num sentido unidirecional, das *proxies* para o NAS, a interface de rede 1 (IR 1) pode ser configurada para que apenas possa receber pacotes. O mesmo se aplica para a interface de rede 2 (IR 2), de modo a que apenas possa enviar pacotes.

4.4 Descrição do sistema

O comportamento do sistema pode ser dividido em cinco partes. Cada uma dessas partes diz respeito ao papel de um dos componentes do sistema. Como a comunicação no sistema é unidirecional, ou seja, a informação flui apenas num sentido, torna-se simples e de fácil entendimento. Na Figura 4.8 estão numeradas, de 1 a 5, as fases do serviço disponibilizado do sistema com numeração.

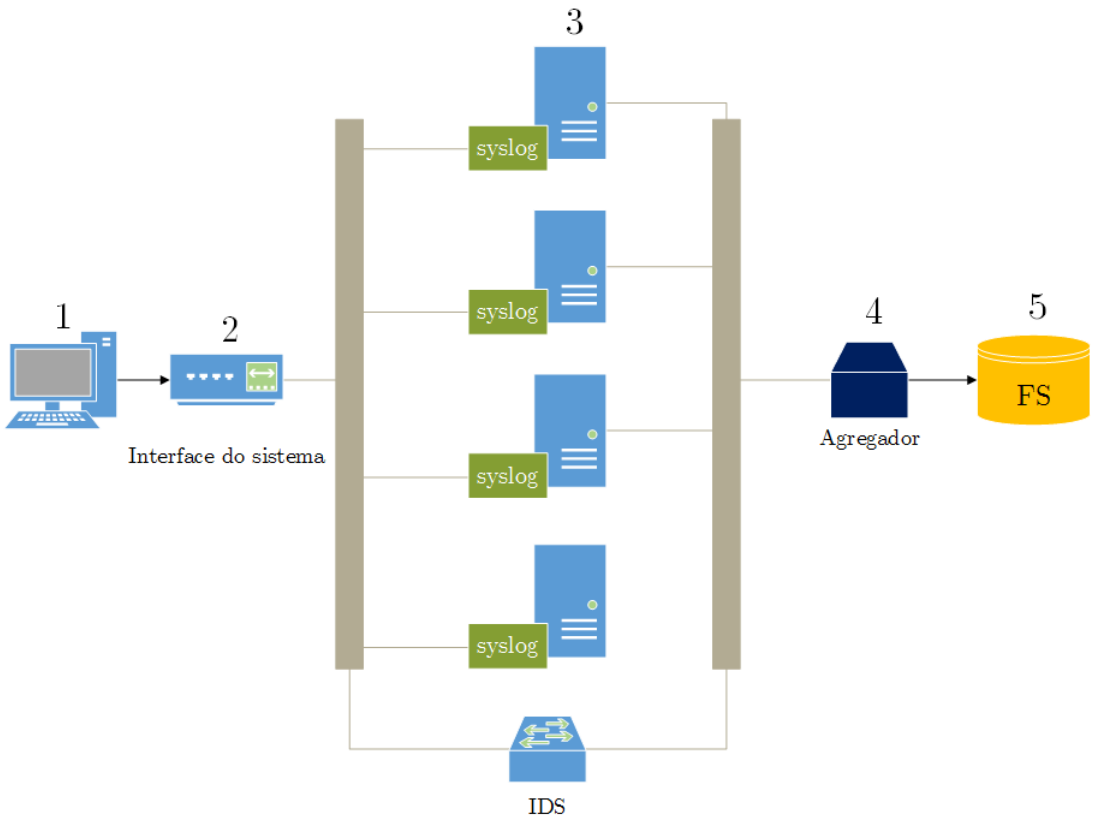

Figura 4.8: Fases do serviço disponibilizado pelo sistema.

Na primeira fase, o produtor de *logs* utiliza o *daemon* local (`syslogd` ou `klogd`) para fazer um registo. Por sua vez, o *daemon*, encaminha o registo para o endereço IP da *proxy*. Como existem várias *proxies* o DNS pode facultar endereços IP seguindo um esquema *Round Robin*.

Na segunda fase, o *log* chega até uma das *proxies* do sistema. O algoritmo da *proxy* está descrito no Algoritmo 1. A *proxy* recebe o *log* e cifra o seu conteúdo (linhas 8 e 11). Depois de cifrado o conteúdo, é adicionado um identificador único ao *log*, obtido através do GIU (linhas 10 e 11). De seguida, adiciona o *log* a um *buffer* (linha 12). Se o *buffer* atingiu o número limite de registos, a *proxy* envia-o para os servidores `syslog` (linhas 15 e 16). Caso contrário, aguarda pelo próximo registo. Cada um destes passos envolve a gravação em disco do estado do *log* em questão (linhas 9 e 13). Não se garante que não sejam perdidos registos quando uma *proxy* falha, mas mitiga-se a perda de registos. No caso da *proxy* falhar, quando esta volta ao funcionamento (linhas 1 a 6), verifica o estado o *buffer* e recupera o estado anterior (linhas 18 a 27). Se o *buffer* atingiu número limite de *logs*, envia-o para os servidores `syslog`, caso contrário aguarda pelo próximo *log*.

O algoritmo da terceira fase está descrito no Algoritmo 2. Nesta fase, os servidores `syslog` recebem um *buffer* de *logs* começam o tratamento de cada um deles (linhas 5 a 8). Cada *log* entregue é tratado de acordo com o standard `syslog`[12] (linha 7). Isto envolve níveis de prioridade, tipos de *log*, entre outros parâmetros. Quando terminada a tarefa do `syslog`, o registo é encaminhado para o agregador (linhas 10 e 11). Este processo é replicado pelos $2f + 1 + k$ servidores.

O algoritmo do agregador está descrito no Algoritmo 3. Quando o agregador recebe uma mensagem, começa a quarta fase (linhas 7 a 10). De acordo com o identificador da mensagem, guarda-a numa tabela de dispersão (linha 9). Cada vez que o agregador recebe uma mensagem verifica se já recebeu $f + 1$ mensagens iguais, se sim envia o *log* para o NAS, caso contrário aguarda pela próxima mensagem (linhas 12 e 14). Todas as mensagens repetidas de um *log* que já tinha sido armazenado são descartadas (linha 8). Para tal é mantido um histórico das mensagens recebidas (linha 9).

Na quinta e última fase, o NAS recebe um pedido de escrita de um *log* num ficheiro, de acordo com o tipo de *log*. O *log* está armazenado e fora do alcance dos servidores e dos produtores. Uma vez o *log* escrito no disco WORM, não pode ser apagado.

O NIDS em nada afeta o normal funcionamento dos restantes componentes do sistema, i.e., é um agente passivo. Este vai recolhendo a informação que passa pelas redes do sistema e guardando estatísticas, verificando se a informação está a fluir de forma unidirecional, entre outros aspetos. Quando uma máquina se destaca estatisticamente das outras por fatores consideráveis, por exemplo, enviou mais 10% de tráfego do que as outras, o NIDS entra em ação e agenda a reinicialização dessa réplica. Em qualquer momento, nunca existirão mais do que k máquinas em recuperação. Cada vez que uma máquina

Algoritmo 1 Algoritmo da *proxy*

Utiliza: GIU; TPM criptográfico; Rede;

▷ Início da *proxy*

upon event ⟨*proxy, Iniciar*⟩ **do**
  if $estadoguardado$ **then**
    trigger⟨$recuperarestado$⟩;
  else
    $estado := \emptyset$;
  end if

▷ Quando recebe um *log*

upon event ⟨*Rede* | log⟩ **do**
  trigger⟨$guardar(log)$⟩;
  $id := GIU.proximoid()$;
  $logfinal := [GIU.cifra(log) + id]$;
  $state := state \cap logcifrado$;
  trigger⟨$guardar(state \cap logcifrado)$⟩;

upon exists $|state| > 1000$ **do**
  trigger⟨$Rede, enviar|logs$⟩;

▷ Recuperar estado guardado

procedure $recuperarestado$
  $estado := lerdisco(estado)$;
  $log := lerdisco(log)$;
  if $log \notin estado$ **then**
    $id := GIU.proximoid()$;
    $logfinal := [GIU.cifra(log) + id]$;
    $state := state \cap logcifrado$;
    trigger⟨$guardar(state \cap logcifrado)$⟩;
  end if
end procedure

▷ Guardar dados em disco

procedure $guardar$(dados)
  $escreverdisco(dados)$;
end procedure

Algoritmo 2 Algoritmo do servidor `syslog`

Utiliza: Implementação `syslog`; Rede;

▷ Início do servidor

1: **upon event** ⟨*servidor*, *Iniciar*⟩ **do**
2: **trigger**⟨$recuperarestado$⟩;
3: $syslog := instância($`syslog`$)$;
4:

▷ Quando recebe um conjunto de *logs*

5: **upon event** ⟨*Rede* | *logs*⟩ **do**
6: **for** $log \in logs$ **do**
7: **trigger**⟨ `syslog` |log⟩;
8: **end for**
9:

▷ Log tratado é enviado para o agregador

10: **upon event** ⟨`syslog`, *log*⟩ **do**
11: **trigger**⟨$Rede, enviar|logs$⟩;
12:

Algoritmo 3 Algoritmo do agregador

Utiliza: NAS; Rede;

▷ Início do agregador

1: **upon event** ⟨*agregador*, *Iniciar*⟩ **do**
2: $logs := \emptyset$;
3: **for all** log **do**
4: $mensagem[log.id] := \emptyset$;
5: **end for**
6:

▷ Quando recebe um de *log*

7: **upon event** ⟨*Rede* | *servidor, log*⟩ **do**
8: **if** $log \notin logs$ **then**
9: $mensagem[log.id] := mensagem[log.id] \cap [log, servidor]$;
10: **end if**
11:

▷ Quando o agregador recebeu $f + 1$ mensagens iguais

12: **upon exists** $\exists log \in logs : |mensagem[log]| \geq f + 1$ **do**
13: $NAS.write(log)$;
14: $logs := logs \cap log$;
15:

é reinicializada, arranca com um sistema operativo e uma implementação diferente do `syslog` diferentes das anteriores. Isto torna mais difícil que o atacante comprometa a máquina novamente.

4.5 Caraterização dos componentes

4.5.1 *Proxy*

A *proxy* é a interface do sistema para o exterior. O seu papel é receber os *logs* enviados pelos produtores e encaminhá-los para os consumidores - servidores `syslog`. Para cada *log* recebido pelas *proxies*, estas cifram o seu conteúdo e adicionam-lhe um identificador único obtido através do GIU.

A *proxy* é um componente chave no sistema por duas razões. Em primeiro lugar, se num dado instante a *proxy* for comprometida, todo o conteúdo dos *logs*, posteriores a esse instante, é comprometido. Em segundo lugar, é um ponto crítico do sistema, ou seja, se este falhar, o sistema deixa de disponibilizar o serviço a que se propõe.

A *proxy* consiste numa pequena quantidade de código, que apenas recebe os *logs*, através de uma camada de transporte seguro (TLS) e dissemina-os pelas réplicas do sistema. Idealmente, esta *proxy* era um *hub*, mas como é utilizada uma camada de transporte seguro (ligações ponto-a-ponto), tal não é possível.

Para que a *proxy* não seja um ponto crítico do sistema, esta é replicada. Como cada *proxy* é independente das outras, os seus estados também são independentes.

As *proxies* têm um sistema operativo mais seguro do que os de uso comum. Para tal pode ser utilizado um SELinux[23], por exemplo. Este tipo de sistema operativo é verificado e apenas contém os módulos essenciais, diminuindo assim o número de vulnerabilidades.

Para barrar toda a informação que não seja direcionada ao porto 514 (porto do `syslog`) é utilizado o módulo `iptables`. As *proxies* vão bloquear todo o tráfego que não seja direcionado para o porto 514, como por exemplo, toda as tentativas de mapeamento de portas. O `iptables` é um módulo seguro cujas últimas vulnerabilidades datam de 2001.[1]

[1] `http://www.cvedetails.com/vulnerability-list/vendor_id-959/product_id-1656/Netfilter-Core-Team-Iptables.html`

4.5.2 Servidor `syslog`

O papel dos servidores `syslog` é receber os *logs* dos produtores, tratá-los e encaminhá-los para armazenamento. Quando um servidor recebe um *buffer* de registos, trata-os individualmente, de acordo com o standard `syslog`[12]. Após isso, o *log* é encaminhado para o ficheiro adequado no armazenamento. O armazenamento também segue o standard `syslog`.

Para evitar falhas comuns entre os servidores, cada servidor tem um sistema operativo diferente, variando entre Windows, Linux, Mac OS e Solaris. Além disso serão utilizadas versões diferentes do `syslog`, por sua vez implementadas em linguagens diferentes desde Java, C e Python. Desta forma, reduz-se a possibilidade de uma vulnerabilidade comum que comprometa mais do que f réplicas.

Nos servidores encontra-se a grande parte aplicacional, e como tal existe uma extensão de código considerável, o que se traduz em cerca de quinze a vinte mil linhas de código. Com tal quantidade de código é fácil haver vulnerabilidades. Portanto, é preciso um agregador (Secção 4.5.3) de confiança entre os servidores `syslog` e o NAS, de forma a isolar estas partes uma da outra. Se os servidores assumem um comportamento Bizantino, é altamente desaconselhável que possam aceder ao armazenamento. Assim, o agregador limita o acesso ao NAS através de uma interface simples e rígida.

4.5.3 Agregador

Este componente é de baixa complexidade, isso traduz-se em cerca de duzentas a trezentas linhas de código, o que permite que seja verificado de forma a atestar a sua segurança. O agregador é uma pequena TCB que recebe mensagens dos servidores e quando recebe $f + 1$ cópias de uma mensagem, encaminha apenas uma cópia para o NAS. Para facilitar o processo de comparação de mensagens o agregador faz uso do identificador único de cada mensagem obtido através do GIU.

A TCB é também um ponto único de falha, como tal deve ser replicada para assegurar o serviço a longo termo.

4.5.4 NIDS

O NIDS é uma máquina que apenas monitoriza o tráfego que passa pelas redes do sistema, mais especificamente a rede entre as *proxies* e os servidores e a rede entre os servidores e o agregador. O NIDS vai atentar para o volume de tráfego, número de mensagens, os destinatários das mensagens, o tipo de tráfego, de forma a identificar possíveis réplicas comprometidas. Quando uma das réplicas diverge de uma forma considerável das outras, por exemplo, enviou mais 5% de mensagens em comparação com as outras, é um sinal de um possível comprometimento dessa réplica. Então o NIDS reinicia essa máquina, caso não ultrapasse o limite de k máquinas em recuperação. O NIDS vai atuar de forma análoga sobre máquinas que possuam, por exemplo, 5% menos mensagens, que tentem enviar pacotes para as *proxies*, etc., ou seja, atua quando detetar padrões de atividade que não pertencem ao comportamento normal da máquina.

Além disso, o NIDS tem mecanismos para quando suspeitar de uma réplica ter capacidade de a reiniciar. Após reiniciar, essa réplica tem um sistema operativo e implementação do `syslog` diferentes dos que tinha anteriormente[11]. Desta forma, dificulta-se futuros ataques a essa máquina, utilizando as mesmas técnicas.

O NIDS tem de ter um sistema operativo semelhante ao das *proxies*, i.e., tem um sistema operativo mais seguro do que os de uso comum, como o SELinux[23], por exemplo.

4.5.5 NAS

O papel do NAS é providenciar armazenamento para os registos dos produtores, previamente tratados pelos servidores.

O NAS é um componente que apenas está conectado ao agregador e, portanto, está completamente isolado do restante sistema. Para garantir que, depois de escritos, os *logs* não podem ser alterados, o NAS tem discos WORM.

É sabido que os NASs têm grandes quantidades de código, o FreeNAS[10] tem cerca de cento e setenta mil linhas de código, torna-se difícil de garantir que não têm vulnerabilidades. Contudo, como este apenas se encontra ligado ao agregador, que é uma TCB verificada e atestada com uma interface simples e rígida, não existe perigo de comprometimento dos *logs*.

Capítulo 5

Implementação

Neste capítulo será descrita a concretização do prototipo desenvolvido. Esta descrição passa pela apresentação da sua arquitetura e dos detalhes da concretização. Por forma a melhor ilustrar esta descrição, são apresentadas figuras com uma configuração básica do sistema, composta por: uma *proxy*, um agregador e $2f + 1 + k$ servidores `syslog` com $f = k = 1$.

5.1 Concretização através de virtualização

A concretização do sistema tem como base a abstração de máquinas virtuais e está ilustrada na Figura 5.1. Sobre o hardware está um *hypervisor*, neste caso o Xen[39], que providencia uma plataforma virtual e gere a execução das máquinas virtuais. Além disso, o Xen garante o isolamento entre máquinas. Todos os TPMs podem ser concretizados virtualmente [4], contudo, devido a escassez de tempo não foi possível utilizar tal técnica.

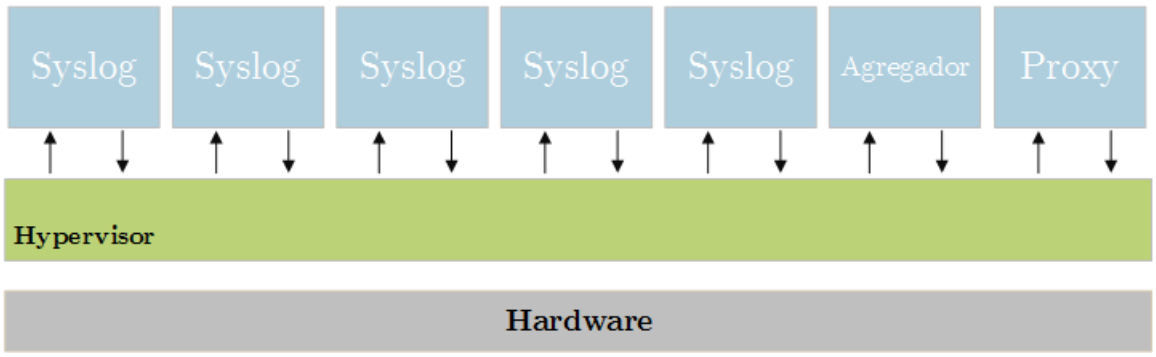

Figura 5.1: Arquitetura do sistema concretizado através de máquinas virtuais.

Ao utilizar-se virtualização, além de se obter uma solução mais barata, torna-se possível concretizar um sistema seguro de *logs* numa única "caixa".

O isolamento das redes é realizada da mesma forma, ou seja, ao invés de serem criadas redes físicas, foram criadas redes virtuais para isolar os componentes que não devem comunicar, como a *proxy* e o NAS, por exemplo.

5.2 Configuração da rede

Na Figura 4.7 estão representadas quatro redes. Três dessas redes (Rede 2, 3, 4) fazem parte do sistema de *logs* e a outra rede pertence ao ambiente onde o sistema se encontra inserido (Rede 1). As redes 2, 3 e 4 são virtuais. Na Tabela 5.1 está descrita a atribuição de endereços IP estáticos às interfaces de rede de cada máquina.

Distribuição de endereços IP estáticos	
Rede 1	
Proxy	Definido pela política da rede
Rede 2	
Proxy	192.168.14.10
Servidor `syslog` 1	192.168.14.20
Servidor `syslog` 2	192.168.14.21
Servidor `syslog` 3	192.168.14.22
Servidor `syslog` 4	192.168.14.23
Rede 3	
Agregador	192.168.214.10
Servidor `syslog` 1	192.168.214.20
Servidor `syslog` 2	192.168.214.21
Servidor `syslog` 3	192.168.214.22
Servidor `syslog` 4	192.168.214.23
Rede 4	
Agregador	192.168.10.10
NAS	192.168.10.20

Tabela 5.1: Distribuição de endereços IP pelas interfaces de rede do sistema.

5.3 Detalhes de implementação

5.3.1 NIDS

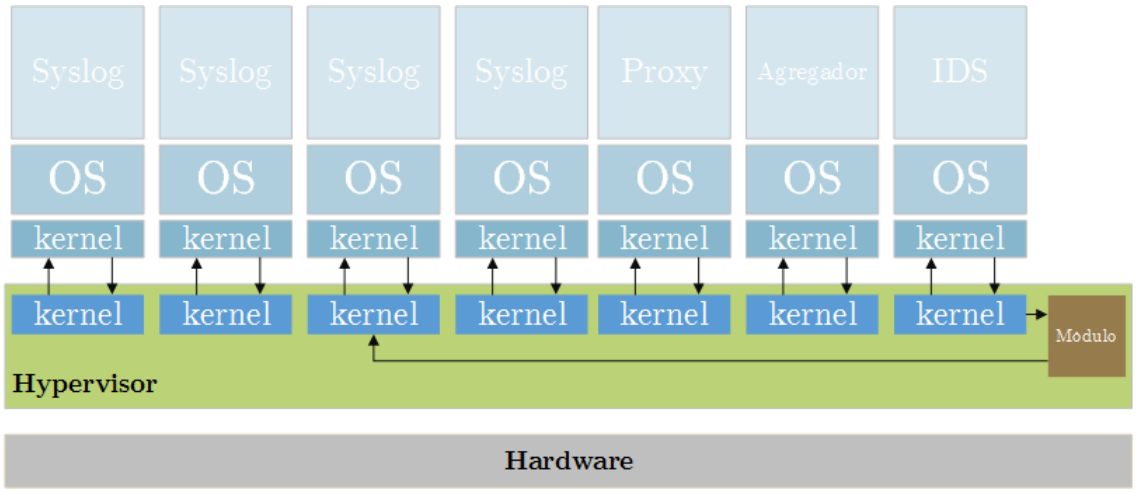

Figura 5.2: Concretização dos mecanismos de controlo do NIDS sobre as máquinas virtuais.

Cada serviço do sistema está a executar numa máquina virtual. Portanto, sendo o *hypervisor* seguro, obtém-se um controlo centralizado e preciso sobre todas as máquinas. Na Figura 5.2 está ilustrada a concretização do mecanismo que o NIDS utiliza para reiniciar as máquinas. Esse mecanismo é um módulo no *hypervisor* que pode ser invocado através do *kernel* do NIDS. O módulo tem os comandos básicos para desligar e ligar máquinas virtuais.

O NIDS foi concretizado com o `Snort`[26]. Para a monitorização de pacotes que se pretende realizar, i.e., contagem de pacotes e verificação de sentidos de tráfego, o `Snort` apresenta todas as capacidades necessárias. O `Snort` permite analisar os pacotes de rede e inserir regras de monitorização, que permitem criar um protótipo funcional. Para que o `Snort` atuasse como pretendido, foram criadas regras de acordo com o descrito no Capítulo 4. Em concreto, as regras são:

```
alert tcp 192.168.14.20 any -> 192.168.14.10 any (message:
  "Tentativa de envio de dados TCP em sentido contrário:
            192.168.14.20 -> 192.168.14.10")

alert tcp 192.168.14.21 any -> 192.168.14.10 any (message:
  "Tentativa de envio de dados TCP em sentido contrário:
            192.168.14.21 -> 192.168.14.10")
```

```
alert tcp 192.168.14.22 any -> 192.168.14.10 any (message:
   "Tentativa de envio de dados TCP em sentido contrário:
              192.168.14.22 -> 192.168.14.10")
alert tcp 192.168.14.23] any -> 192.168.14.10 any (message:
   "Tentativa de envio de dados TCP em sentido contrário:
              192.168.14.23 -> 192.168.14.10")
```

Estas regras monitorizam se algum dos servidores tenta enviar dados, através do protocolo TCP, para a *proxy*. Poder-se-ia definir um conjunto maior de regras, por exemplo, o volume de tráfego ser monitorizado com recurso a contadores. Esta concretização não é perfeita, mas como se trata apenas de um protótipo, é adequada.

Quando uma das regras dispara, é gerado um *log* de controlo. Sempre que o ficheiro de *log* for alterado, é sinónimo que aconteceu algo no sistema que fez disparar uma ou mais regras. O disparo de uma ou mais regras denuncia um ou mais acontecimentos suspeitos. Isto significa que através da análise do *log* podemos agir sobre as réplicas.

Para reiniciar máquinas, utiliza-se um agente em Java que analisa cada alteração do *log* do `Snort` e, caso seja necessário, através do módulo no *hypervisor*, solicita a execução dos seguintes comandos: `xm destroy OSY` para forçar a máquina a desligar e `xm start OSZ` para ligar uma nova máquina.

5.3.2 *Proxy*

A *proxy* é uma pequena porção de código em Java para receber os pacotes, colocar o identificador obtido através do GIU e encaminhá-los para os servidores. O identificador de cada *log* é colocado no início do conteúdo de cada registo. Deste modo, quando os servidores tratarem o *log* sabem que os primeiros bytes dizem respeito ao identificador. Para beneficiar o desempenho do sistema, as *proxies* utilizam *batching*, i.e., enquanto receberem dados vindos da rede é criado localmente um *buffer* de *logs*. Cada *log* recebido é guardado em disco e em memória. Quando o número de *logs* atingir um certo limiar, a *proxy* envia o *buffer* de registos de uma vez só. No caso da *proxy* falhar, quando for recolocada em funcionamento, lê o estado guardado em disco e continua a disponibilizar o serviço normalmente e a perda de registos é mitigada.

`iptables`

A *proxy* tem um módulo `iptables` para filtrar pacotes que não sejam *logs* e rejeitar qualquer pacote vindo dos servidores `syslog`. Bastam duas regras para realizar tal filtro: uma regra que aceita os pacotes direcionados ao porto 514 e uma regra que rejeita todos os restantes pacotes. Mais especificamente as regras são:

- Para a interface de rede ligada à rede externa (Figura 4.7 Rede 1)

  ```
  iptables -A INPUT -s 192.168.0.0/24 -p tcp --dport 514
                          -j ACCEPT
  ```

  Esta regra permite conexões TCP no porto 514 vindas da rede 192.168.0.0/24. Esta máscara de sub-rede teria de ser configurada de acordo com a rede onde o sistema seria colocado;

- Para ambas as interfaces de rede (Figura 4.7 Rede 1 e 2)

  ```
  iptables -P INPUT DROP
  ```

  Esta regra define *DROP* como a política por omissão na cadeia *INPUT*. Isto significa que, se um pacote recebido pela máquina não se enquadrar numa das regras anteriores, é descartado.

Gerador de Identificadores Únicos (GIU)

O GIU devia ser implementado sob forma de uma TPM, contudo não foi possível ter acesso a um módulo desses. Assim, o GIU foi concretizado através de dois métodos escritos em Java. Um método para criar um identificador único e outro método para cifrar o conteúdo de *logs*. Como apenas se pretendia construir um protótipo, esta aproximação foi suficiente.

5.3.3 NAS

O NAS é um sistema separado do sistema de *logs* e está completamente isolado, apenas tendo uma ligação de rede com o agregador. No sistema foi utilizado o FreeNAS[10], que para além de gratuito é completo e é *open source*.

5.3.4 Servidores

Os servidores têm diferentes sistemas operativos e correm servidores `syslog` implementados em diferentes linguagens. Existem i imagens de máquinas virtuais com sistemas operativos diferentes, com $i = n + 1$ e $n = 2f + 1$. Desta forma, quando uma máquina é reinicializada recebe um novo sistema operativo que não estava em utilização anteriormente por nenhuma das outras réplicas. Para essa variedade de imagens utilizou-se: Mac OS, Windows, Solaris e Linux (CentOS).

5.4 Fluxo de dados

Nas Figuras 5.3, 5.4 e 5.5 está ilustrado o fluxo de dados para o tratamento de um *log*, desde o momento que é recebido pela *proxy* até chegar ao agregador. Todas as máquinas têm duas interfaces de rede, exceto o NAS, que neste caso, é um sistema separado. Existem três fases de comunicação distintas, cada uma delas está representada numa figura diferente.

A primeira fase de comunicação está ilustrada na Figura 5.3, essa fase começa quando um *log* é enviado para o sistema. A máquina física recebe essa mensagem na sua interface de rede física e encaminha-a para a *proxy*, que por sua vez, recebe esse *log* na sua interface de rede virtual 2 (IRV 2). Para enviar a mensagem para os servidores é utilizada a IRV 1. Todos os servidores e o NIDS recebem esse pacote na IRV 1.

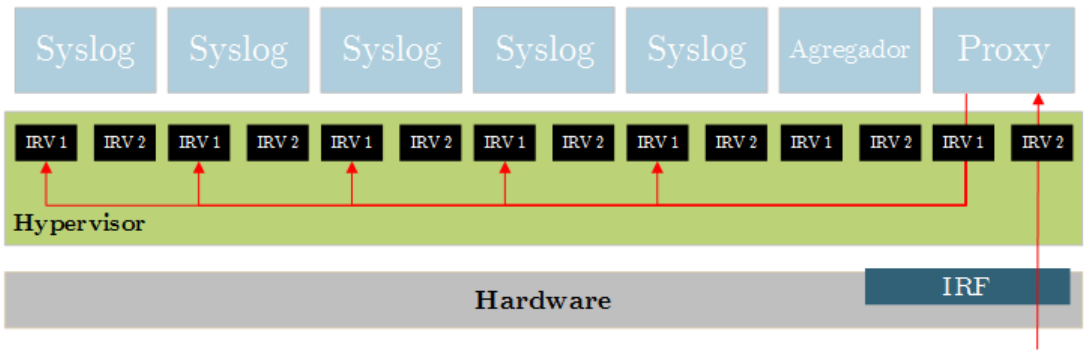

Figura 5.3: Fluxo de dados entre a *proxy* e os servidores.
MV - Máquina Virtual; IRV - Interface de Rede Virtual; IRF - Interface de Rede Física

Após os servidores tratarem o *log* começa a segunda fase de comunicação. Esta fase está ilustrada na Figura 5.4. Os servidores utilizam a IRV 2 para enviar o *log* para o agregador. Tanto o agregador como o NIDS recebem essa mensagem na IRV 1 e na IRV 2, respetivamente.

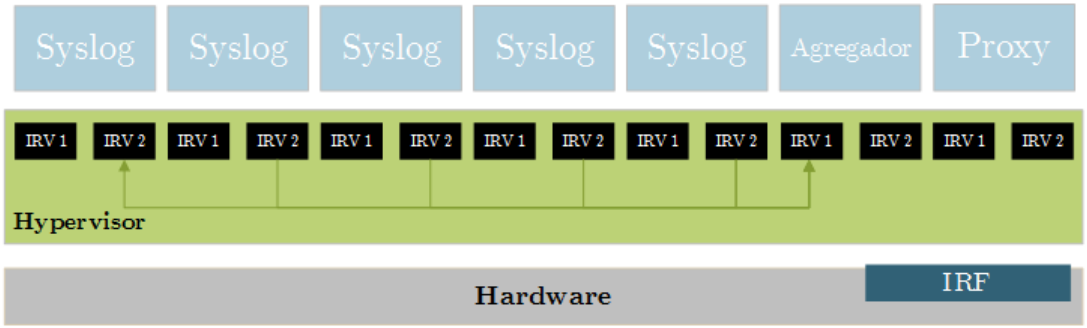

Figura 5.4: Fluxo de dados de dados entre os servidores e o agregador.
MV - Máquina Virtual; IRV - Interface de Rede Virtual; IRF - Interface de Rede Física

Quando o agregador recebe mensagens suficientes para encaminhar o *log* para o NAS, dá-se início à terceira fase de comunicação, que está representada na Figura 5.5. O agregador utiliza a IRV 2 para enviar o *log* para armazenamento. O NAS, sendo um sistema à parte, recebe a mensagem na sua IRF.

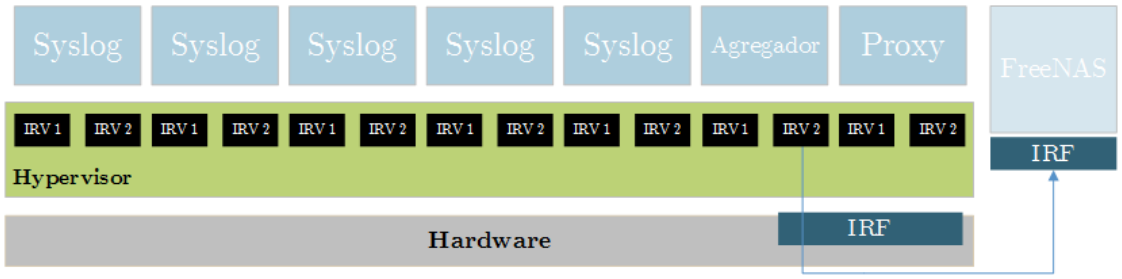

Figura 5.5: Fluxo de dados entre o agregador e o NAS.
MV - Máquina Virtual; IRV - Interface de Rede Virtual; IRF - Interface de Rede Física

Para cada registo enviado pelos produtores, este é o caminho que tem de percorrer até ser armazenado.

Capítulo 6

Resultados

Neste capítulo são apresentados os resultados obtidos com o protótipo desenvolvido. Para esses resultados contribuíram: o ambiente onde foram feitos os testes ao protótipo, o hardware utilizado e os casos testados.

6.1 Protótipo

O protótipo foi desenvolvido utilizando a linguagem de programação Java 7. O protótipo não ficou completo, uma vez que ficaram por implementar bastantes aspetos como: o GIU sob forma de um TPM; um conjunto de regras completo para o NIDS, entre outros. Para se implementar o GIU utilizou-se uma pequena função criptográfica e o NIDS apenas possuía uma regra no conjunto de regras.

Além de não estar completo, tem alguns aspetos de implementação não terminados como: não suportar registos com mais de 1024 bytes, por alguma razão não identificada; a comunicação entre os servidores e o agregador ser com o *Remote Method Invocation* (RMI) do Java; a restante comunicação do sistema foi feita com base no protocolo de transporte UDP; entre outros.

O protótipo é rudimentar, no entanto é suficiente para oferecer uma noção do funcionamento e desempenho do sistema. Desta forma, foi possível avaliar alguns pontos, marcar outros que precisam de ser melhorados e aprovar ideias que, até então, apenas estavam no papel.

O protótipo foi executado numa máquina com as seguintes especificações:

- Processador: Intel (R) Core (TM) 2 CPU 6400 @ 2.13GHz;
- Memória: 6Gb DDR 2 800MHz;
- Placa de Rede: NetXtreme BCM5754 Gigabit Ethernet PCI Express;
- Disco: 160Gb SATA.

Nesta máquina foi instalado um servidor *Citrix XenServer*.

O protótipo testado tem uma configuração básica do sistema, composta por: uma *proxy*, um agregador, um FreeNAS e $2f + 1$ servidores `syslog`, com $f = 1$. O protótipo foi executado em máquinas virtuais criadas no servidor. Essas máquinas virtuais tinham as seguintes especificações:

- 1 processador;
- 256Mb DDR 2 800MHz de memória;
- 2 placas de rede;
- 20Gb de armazenamento.

Cada máquina virtual tinha duas placas de rede, de modo a haver uma separação explícita das redes a que estavam ligadas.

6.2 Testes

Os testes realizados incidiram no desempenho do sistema. Foram realizadas várias séries de testes. Cada bateria de testes visava um sistema que produzia registos de 128, 256 ou 512 bytes e os tempos foram medidos em segundos.

Primeiro, testou-se o `syslog` normal, ou seja, foi utilizada uma implementação do `syslog` em Java, e medidos os tempos para tratar 25000 pedidos para cada tamanho de registo. Na Figura 6.1 estão representados a azul os tempos obtidos.

Em segundo, testou-se o protótipo sem *batching*. A linha de teste foi exatamente a mesma, tratar 25000 pedidos para cada tamanho registo. Os tempos obtidos estão representados a laranja na Figura 6.1.

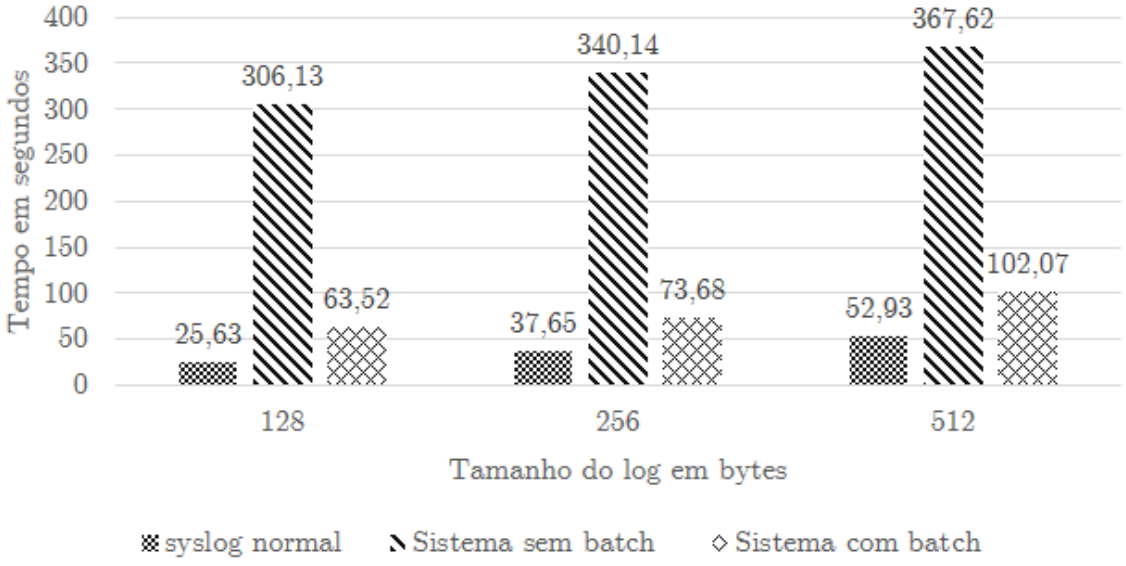

Figura 6.1: Gráfico dos tempos dos testes realizados.

Por fim, testou-se o prótipo com *batching*. A linha de teste foi igual à anterior, com um limiar de *batching* de mil registos. Estão representados a cinzento, na Figura 6.1, os tempos obtidos.

Na Figura 6.2 estão apresentados os mesmos dados da Figura 6.1, mas em termos de pedidos executados por segundo.

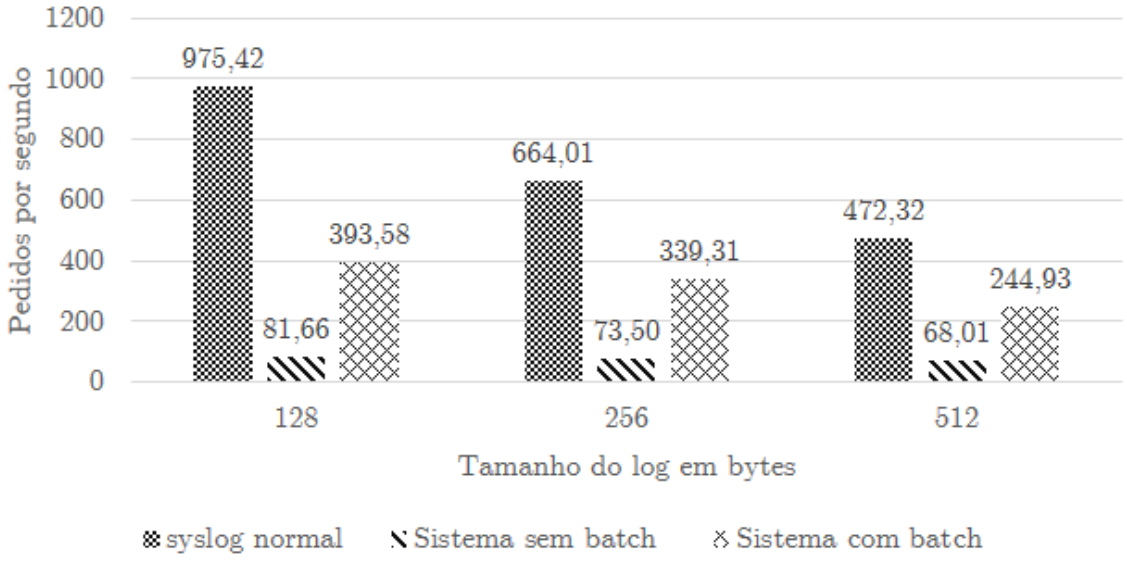

Figura 6.2: Gráfico do número de pedidos tratados por segundos dos sistemas testados.

Na Tabela 6.1 estão representados os tempos de todos os testes. Para além disso, estão representadas as percentagens em relação ao tempo do `syslog` normal.

Comparação de tempos			
Mensagens	`syslog` normal	Sistema sem *batch*	Sistema com *batch*
128	25,63	306,13 (+1094%)	63,52 (+148%)
256	37,65	340,14 (+803%)	73,68 (+96%)
512	52,93	367,62 (+594%)	102,07 (+93%)

Tabela 6.1: Comparação de tempos, para 25000 pedidos, com diferentes tamanhos de mensagem.

Poder-se-ia definir um conjunto maior de testes, incluindo outros tais como:

- Testes ao sistema diferentes números de réplicas;
- Testes ao desempenho enquanto uma réplica é reiniciada;
- Tentativas de comprometer os registos;
- Comprometer um servidor e atacar os outros, de forma a testar a reação do NIDS;
- Testar possíveis vulnerabilidades em cada máquina.

Contudo, devido a limitações de tempo neste projeto e de hardware, não foi possível realizar todos os testes pretendidos, sendo que estes farão parte de desenvolvimento de trabalho futuro.

6.3 Discussão dos resultados

Os resultados são os esperados, se a complexidade aumenta, o tempo despendido por pedido aumenta. Os testes ao `syslog` normal foram utilizados como termo de comparação.

Quando se testou o sistema sem *batching*, os tempos obtidos mostram uma diferença considerável em comparação com `syslog` normal. Essa diferença é caracterizada por um fator de aproximadamente dez. Ainda assim, tendo em conta a quantidade de fases que se colocou entre a *proxy* e o NAS, essa diferença faz sentido. Para um registo chegar até ao armazenamento, tem de passar pela rede pelo menos três vezes, o que não acontece no `syslog` normal.

Utilizando *batching*, os tempos obtidos foram consideravelmente diferentes. Como se poupa nas operações de rede - o que, saliente-se, dizem respeito a uma grande porção os tempos recolhidos -, os resultados são cerca do dobro em relação ao `syslog` normal, o que é aceitável.

É também importante mencionar o fator hardware. A máquina em questão é modesta e isso não favorece os resultados. Colocar cinco máquinas virtuais a executar, numa máquina com estas especificações, é sinónimo de concorrência entre elas, por tempo de processamento. Essa concorrência não traz benefícios para os tempos recolhidos. Isto significa, que tendo uma máquina com hardware mais recente, os tempos seriam influenciados.

Capítulo 7

Conclusão

Esta dissertação descreve um sistema de *logs* indelével, isto é, um sistema onde os registos não podem ser apagados. Para além disso, os registos em trânsito e em repouso têm garantias de integridade e confidencialidade. Foi utilizado um modelo híbrido de faltas, uma vez que, os diferentes componentes do sistema têm diferentes níveis de ameaça e complexidade. O sistema é constituído por vários componentes, responsáveis por, em conjunto, garantirem as propriedades propostas. Os registos dos produtores são encaminhados para as *proxies* - interface do sistema. As *proxies* disseminam-os pelos servidores `syslog` que os tratam. Para que apenas seja armazenada uma cópia de cada registo, são utilizados agregadores como mediadores entre os servidores e um NAS. Ao utilizar interfaces simples e rígidas limitou-se a interação entre componentes e com o exterior.

A grande contribuição deste trabalho passa pelo desenvolvimento de um sistema de *logs* que, para além de garantir segurança para os registos em trânsito e em repouso, garante ainda que os *logs* em repouso não podem ser apagados. Até agora nenhuma solução tinha garantido estas três propriedades em simultâneo.

As contribuições deste trabalho, que estão descritas ao longo deste documento, são:

1. **Confidencialidade dos *logs*:** o conteúdo dos registos é protegido desde que saem do produtor, até entrarem no sistema. Após a entrada no sistema, o seu conteúdo é cifrado com uma chave pública e, só o detentor dessa chave, terá acesso ao seu conteúdo;
2. **Integridade dos *logs*:** a integridade dos registos é garantida tanto para os *logs* em trânsito como para os *logs* em repouso;
3. ***logs* indeléveis:** uma vez armazenado um registo, é garantido que não pode ser

apagado ou alterado;

4. **Um sistema com um modelo híbrido de faltas:** os componentes mais complexos do sistema têm mais probabilidade de ter vulnerabilidades, da mesma maneira que se espera que tenham mais paragens e sejam alvos mais fáceis. Todavia, o sistema continua a prestar o serviço, mesmo se algumas partes falharem ou forem comprometidas.

7.1 Trabalho futuro

Para além de uma sessão de testes mais intensiva, existem vários pontos que necessitam de trabalho. Antes de poder ser testado, é necessário realizar uma implementação completa do protótipo. Vários aspetos não ficaram terminados, finalizá-los seria a próxima etapa.

Para melhorar o desempenho do sistema também seria interessante criar instanciações do serviço, ou seja, criar várias réplicas do serviço, dentro da "caixa preta", de forma a aumentar a disponibilidade do sistema.

Outro ponto muito importante é garantir que não se perdem registos nas *proxies*. Para tal, os produtores têm de enviar o *log* para todas *proxies* para que, caso uma *proxy* falhe, não se percam os registos que apenas essa *proxy* estava a tratar. Tal modificação acrescenta complexidade ao sistema, porque existirão mais cópias do registo para filtrar, uma vez que no armazenamento, só estará uma cópia para cada registo.

Abreviaturas

CIFS *Common Internet File System*.

CPU *Central Processing Unit*.

DNS *Domain Name System*.

GIU Gerador de Identificadores Únicos.

HIDS *Host-based Intrusion Detection System*.

HTTP *Hypertext Transfer Protocol*.

IP *Internet Protocol*.

IPv6 *Internet Protocol versão 6*.

iSCSI *Internet Small Computer System Interface*.

LAN *Local Area Network*.

MAC *Message Authentication Code*.

NAS *Network-Attached Storage*.

NAT *Network Address Translation*.

NFS *Network File System*.

NIDS *Network Intrusion Detection System*.

`klogd` `kernel log daemon.`

`syslogd` `syslogd daemon.`

syslog-ng syslog new generation.

RAID *Redundant Array of Independent Drives*.

RFC *Request for Comments*.

RIP *Routing Information Protocol*.

RMI *Remote Method Invocation*.

SCSI *Small Computer System Interface*.

SHA-1 *Secure Hash Algorithm 1*.

SK Schneier-Kelsey.

SSH *Secure Shell*.

SSL *Secure Socket Layer*.

TCB *Trusted Computing Base*.

TCP *Transmission Control Protocol*.

TLS *Transport Layer Security*.

TPM *Trusted Platform Module*.

UDP *User Datagram Protocol*.

WORM *Write Once Read Many*.

Bibliografia

[1] Rafael Accorsi. Safe-keeping digital evidence with secure logging protocols: State of the art and challenges. In *Proceedings of the 2009 Fifth International Conference on IT Security Incident Management and IT Forensics*, IMF '09, pages 94–110, Washington, DC, USA, 2009. IEEE Computer Society.

[2] Rafael Accorsi. Bbox: a distributed secure log architecture. In *Proceedings of the 7th European conference on Public key infrastructures, services and applications*, EuroPKI'10, pages 109–124, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Mihir Bellare and B Yee. Forward integrity for secure audit logs. 1997.

[4] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: virtualizing the trusted platform module. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[5] B. Callaghan, B. Pawlowski, and P. Staubach. RFC1813: NFS version 3 protocol specification, June 1995. *See also RFC1094 [19]*, 1995.

[6] Bin-Hui Chou, Kohei Tatara, Taketoshi Sakuraba, Yoshiaki Hori, and Kouichi Sakurai. A Secure Virtualized Logging Scheme for Digital Forensics in Comparison with Kernel Module Approach. *2008 International Conference on Information Security and Assurance (isa 2008)*, pages 421–426, April 2008.

[7] Inc. D. New, M. Rose, Dover Beach Consulting. RFC3195 Reliable Delivery for syslog. 2001.

[8] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[9] Dario V. Forte, Cristiano Maruti, Michele R. Vetturi, and Michele Zambelli. Secsyslog: an approach to secure logging based on covert channels. In *Proceedings of the*

First International Workshop on Systematic Approaches to Digital Forensic Engineering on Systematic Approaches to Digital Forensic Engineering, SADFE '05, pages 248–, Washington, DC, USA, 2005. IEEE Computer Society.

[10] FreeNAS. http://www.freenas.org/, consultado pela última vez em: 21 setembro de 2013.

[11] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Os diversity for intrusion tolerance: Myth or reality? In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, DSN '11, pages 383–394, Washington, DC, USA, 2011. IEEE Computer Society.

[12] R Gerhards. RFC5424 The Syslog Protocol. 2009.

[13] Cisco Systems J. Kelsey, NIST, J. Callas, PGP Corporation, A. Clemm. RFC5424 Signed syslog Messages. 2009.

[14] Nobutaka Kawaguchi, Shintaro Ueda, and Naohiro Obata. A secure logging scheme for Forensic Computing. *Information Assurance Workshop, Proceedings from the Fifth Annual IEEE SMC*, pages 386–393, 2004.

[15] Karen Kent and Murugiah P. Souppaya. Sp 800-92. guide to computer security log management. Technical report, Gaithersburg, MD, United States, 2006.

[16] Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5(1):2:1–2:21, March 2009.

[17] Paulo Jorge Sousa Miguel Pupo Correia. *Segurança no Software*. 1° edição edition, 2010.

[18] (Microsoft) Paul J. Leach and (Microsoft) Dilip C. Naik. Internet Draft - A Common Internet File System (CIFS/1.0) Protocol. 1997.

[19] I Ra and TK Park. A FORENSIC LOGGING SYSTEM BASED ON A SECURE OS. *International Journal of Computer Science and Applications*, 6(3):75–91, 2009.

[20] Roi Saltzman. Active Man in the Middle Attacks A whitepaper from IBM Rational Application Security Group. 2009.

[21] J. Satran, K. Meth, IBM, C. Sapuntzakis, Cisco Systems, M. Chadalapaka, Hewlett-Packard Co., and E. Zeidner. RFC3720 Internet Small Computer Systems Interface (iSCSI). 2004.

[22] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, May 1999.

[23] SELinux. http://www.nsa.gov/research/selinux/, consultado pela última vez em: 21 setembro de 2013.

[24] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[25] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[26] Snort. http://www.snort.org/, consultado pela última vez em: 21 setembro de 2013.

[27] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. Resilient intrusion tolerance through proactive and reactive recovery. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, PRDC '07, pages 373–380, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Udo Steinberg and Bernhard Kauer. NOVA: a microhypervisor-based secure virtualization architecture. *Proceedings of the 5th European conference on Computer systems*, EuroSys:209–222, 2010.

[29] Syslog. http//www.syslog.org/, consultado pela última vez em: 21 setembro de 2013.

[30] Syslog-ng. http://www.balabit.com/network-security/syslog-ng/, consultado pela última vez em: 21 setembro de 2013.

[31] Inc. T. Dierks, Independent, E. Rescorla, RTFM. RFC5246 The Transport Layer Security (TLS) Protocol Version 1.2. 2008.

[32] T.N. Newsham T. Ptacek. Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks Inc., 1998.

[33] James Turnbull. Understanding Logging and Log Monitoring. In *Hardening Linux*, chapter 9. Apress, 2005.

[34] Paulo Verissimo and Luis Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[35] Paulo E. Veríssimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81, March 2006.

[36] VMware. http://www.vmware.com/, consultado pela última vez em: 21 setembro de 2013.

[37] Zhi Wang and Xuxian Jiang. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity. *2010 IEEE Symposium on Security and Privacy*, pages 380–395, 2010.

[38] ME Whitman and HJ Mattord. *Principles of information security*. Course Technology, 2010.

[39] Xen. http://www.xenproject.org/, consultado pela última vez em: 21 setembro de 2013.

[40] André Zúquete. *Segurança em redes informáticas*. FCA - Editora de informática, 3° edição edition, 2010.